Security of the pseudorandom number generators and implementations of public key signature schemes

Jean-Christophe Zapalowicz

ANNÉE 2014

THÈSE / UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Jean-Christophe Zapalowicz

préparée au centre de recherche Inria

Co-encadrée par Thomas Jensen, directeur de recherche, Inria

Sécurité des générateurs pseudo-aléatoires et des implémentations de schémas de signature à clé publique

Thèse soutenue à Rennes
le 21 Novembre 2014

devant le jury composé de :

Louis Goubin / rapporteur
Professeur — UVSQ

Emmanuel Prouff / rapporteur
Expert — ANSSI

Christophe Clavier / examinateur
Professeur — Université de Limoges

Sylvain Duquesne / examinateur
Professeur — Université de Rennes 1

Benoît Gérard / examinateur
Expert — DGA

Benjamin Smith / examinateur
Chargé de recherche — Inria

Mehdi Tibouchi / examinateur
Chercheur — NTT Secure Platform Lab.

Pierre-Alain Fouque / co-directeur
Professeur — Université de Rennes 1

Remerciements

Les premières lignes sont souvent les plus difficiles à trouver et, après réflexion, je les réserve pour Aurélie Bauer et Damien Vergnaud. Si j'ai débuté une thèse, c'est en grande partie grâce à grâce à eux. Ils m'ont dévoilé une voie dans la recherche que je ne pensais pas faite pour moi, et je les en remercie. J'en profite pour saluer toutes les personnes de l'ANSSI que j'avais côtoyées durant mon stage et qui ont certainement participé à mon envie d'en apprendre plus : Henri, Thomas et Thomas, Victor, Joana, Eliane, Yannick, Jean-René, Jean-Claude, Karim et Marion. Enfin, je souhaite remercier David Pointcheval qui m'a offert un poste à l'ENS le temps que mon contrat doctoral se concrétise et a financé plusieurs de mes déplacements.

Je remercie Louis Goubin et Emmanuel Prouff qui ont accepté de rapporter ma thèse, et plus généralement tous les membres de mon jury de thèse. Je n'oublie pas bien sûr la DGA qui a financé mes travaux de recherche et Inria qui m'a fourni un environnement de travail remarquable.

J'en arrive à celui envers qui je suis le plus reconnaissant : Pierre-Alain Fouque. Faire une thèse c'est bien, s'épanouir pendant trois années c'est encore mieux et Pierre-Alain y est pour beaucoup. Je le remercie donc pour sa présence, sa motivation, son enthousiasme et son expérience de la recherche qu'il a su me transmettre. J'ai également une pensée particulière pour Mehdi Tibouchi qui, bien que travaillant fort loin d'ici, a également beaucoup rythmé ces trois années de thèse grâce à ses idées et son savoir.

Je souhaite remercier les membres de l'équipe Celtique dans laquelle j'ai évolué pendant un certain temps, et Lydie qui a toujours été présente pour répondre à mes questions administratives. Je remercie également mes anciens et actuels collègues de bureau pour les très bons moments passés ensemble : Delphine et Arnaud suivis de Pierre, Pierre et Brice. Je salue chaleureusement les personnes de la DGA qui ont souvent ponctué mes vendredis et qui ont, pour certains, participé à mes travaux de recherche : Marion, Vivien, Thomas, Raphaël, Benoît et Jean-Gabriel. Enfin je salue le couple madrilène Gilles et François, les anciens thésards Jérémy et Patrick, et les actuels que sont Sonia et Adrian.

Pour finir sur une note beaucoup plus personnelle, je tiens à remercier Astrid qui m'aura attendu pendant ces trois longues années. Cela n'a pas toujours été facile, la distance n'a pas aidé, mais au final, nous sommes toujours ensemble et probablement renforcés. Merci d'avoir été patiente et compréhensive ! C'est aussi grâce à toi que cette thèse s'est bien déroulée !

Table des matières

Chapitre 1
Introduction

La *cryptographie* est l'art de chiffrer un message, c'est-à-dire de l'écrire sous une forme inintelligible pour quiconque ne connaît pas le processus de chiffrement. Associée à la *cryptanalyse* qui peut être vue comme l'art de déceler des faiblesses dans ces processus de chiffrement, ces notions forment les deux pans de la *cryptologie*, la science du secret.

De nombreux exemples d'utilisation de la cryptographie ponctuent l'Histoire, principalement militaire, tel le code de César à l'ère romaine ou bien la machine Enigma durant la Seconde Guerre mondiale. Mais ce n'est qu'à partir de la seconde moitié du XX^{e} siècle que cette science a significativement évolué grâce à une volonté de formalisation, l'avancée des systèmes de communication et le développement de l'informatique. Ainsi, alors que par le passé il était fréquent de garder secret la totalité des processus de chiffrement et de déchiffrement, la cryptographie moderne repose sur le principe de Kerckhoffs énoncé en 1883 selon lequel un schéma cryptographique ne devrait utiliser que des algorithmes publics et seulement une petite information secrète, la *clé*. Deux avantages résultent de ce principe. D'une part une clé secrète compromise peut être remplacée sans avoir à changer le schéma cryptographique en entier. D'autre part, lorsqu'un cryptographe propose un nouveau schéma ou un nouvel algorithme, l'ensemble de la communauté, en particulier les cryptanalystes, peuvent l'étudier et donner du crédit à sa sécurité en fonction de leurs résultats.

Aujourd'hui, la cryptographie n'est absolument plus restreinte au domaine militaire et fait même partie de notre vie quotidienne. Cartes de crédit, téléphones mobiles, réseaux Wifi, navigateurs Internet et passeports sont des exemples de technologies utilisant des algorithmes cryptographiques.

1.1 Cryptographie moderne

1.1.1 Généralités

L'objectif de la cryptographie moderne consiste à fournir diverses fonctionnalités de sécurité pour des communications via un canal non sécurisé comme un téléphone portable ou Internet, c'est-à-dire un canal qui peut être espionné, voir manipulé. La plus naturelle d'entre elles est la *confidentialité* des données : seule une personne avec la bonne clé secrète doit être en mesure de comprendre les messages d'une communication. Mais deux autres fonctionnalités tout aussi importantes pour de nombreuses applications sont offertes par la cryptographie : l'*authentification* qui assure l'identité de l'expéditeur d'un message et l'*intégrité* qui assure qu'un message n'a pas été modifié. Il en existe de nombreuses autres disponibles selon les schémas proposés, on peut citer la *signature électronique* ou le *calcul distribué multi-parties*. Ainsi, le cryptographe développe des schémas qui assurent ces propriétés de sécurité alors que le cryptanalyste recherche des failles impliquant qu'elles ne sont pas vérifiées. Il reste à définir comment mesurer la sécurité de ces schémas.

La cryptographie moderne est basée sur l'*impossibilité calculatoire* : tout schéma cryptographique peut théoriquement être cassé grâce à une recherche exhaustive de la clé secrète, c'est l'attaque par force brute. Il suffit à l'attaquant d'avoir à disposition un message et son chiffré, hypothèse tout à fait réaliste dans la plupart des applications, de lancer l'algorithme de déchiffrement avec toutes les clés possibles et de comparer le résultat du déchiffrement avec le message connu. Cependant, pour une clé secrète codée sur 128 bits le nombre de candidats à tester s'élève à 2^{128}. En d'autres termes, si un attaquant a accès à un milliard d'ordinateurs capables d'effectuer chacun un milliard de déchiffrement par seconde (ce qui est bien bien au-dessus des capacités actuelles), il lui faudrait 1.1×10^{13} années (770 fois l'âge estimé de l'univers) pour effectuer les 2^{128} déchiffrements et retrouver la clé secrète. Ainsi, avec des tailles de clé correctement choisies, la recherche exhaustive est rendue impossible de manière calculatoire.

La cryptographie est divisée en deux branches bien distinctes : la *cryptographie symétrique* également appelée cryptographie à clé secrète, et la *cryptographie à clé publique* aussi dite asymétrique.

1.1.2 Cryptographie symétrique

La cryptographie symétrique est de loin la plus ancienne puisque tous les algorithmes utilisés depuis l'Antiquité jusqu'en 1976 tombent dans cette catégorie. Elle repose sur le principe relativement intuitif que deux personnes souhaitant communiquer de façon sécurisée partagent une clé secrète commune permettant de chiffrer et déchiffrer des messages. Les schémas symétriques sont en général supposés sûrs s'il n'existe aucune attaque plus efficace qu'une attaque par force brute. Cependant, l'absence de ce genre d'attaque ne signifie pas qu'il n'en existe pas. La confiance en la sécurité d'une tel schéma ne peut en conséquence être établie en général qu'après un certain nombre d'années d'analyse par la communauté. Un autre inconvénient de ce type de schéma porte sur le caractère secret de la clé. D'une part, une clé doit être générée pour chaque couple de personnes souhaitant communiquer. Ainsi, si une personne souhaite recevoir des messages de x personnes, il doit avoir en sa possession x clés secrètes. D'autre part, avant d'engager une conversation avec une personne, il faut partager une clé secrète, ce qui requiert soit une rencontre physique soit l'utilisation d'un schéma additionnel.

Les principales primitives, c'est-à-dire les briques de base d'un schéma, utilisées en cryptographie symétrique se répartissent en quatre catégorie. En premier, citons les primitives de *chiffrement par bloc* qui permettent de chiffrer et déchiffrer un bloc de message en utilisant une clé secrète. Pour chiffrer des messages de taille arbitraire, on utilise un mode opératoire qui définit la méthode pour enchaîner les appels à la primitive de chiffrement par bloc. Les primitives de *chiffrement par flot* forme une deuxième catégorie de cette branche. Elles produisent une suite aléatoire à partir d'une clé. Le chiffrement d'un message s'effectue en sommant ce dernier avec la suite obtenue et le déchiffrement est réalisé grâce à l'opération inverse. Les *fonctions de hachage* sont des fonctions publiques qui doivent se comporter comme des fonctions aléatoires et, plus précisément, il doit être difficile de les inverser et de trouver des collisions. Il est important de préciser que ces fonctions ne font intervenir aucun secret même si elles sont classées dans la famille de la cryptographie symétrique du fait de la similarité de leur construction avec celle des deux catégories de primitives précédentes. Les fonctions de hachage sont utilisées dans de très nombreux schémas cryptographiques à clé secrète et à clé publique. Enfin les *codes d'authentification de message* ou MAC permettent d'authentifier l'auteur d'un message et l'intégrité de ce message grâce à une clé secrète. Les MAC les plus répandus sont construits à partir d'une

fonction de hachage ou d'un schéma de chiffrement par bloc.

1.2 Cryptographie à clé publique

En 1976, deux cryptographes américains Whitfield Diffie et Martin Hellman [DH76] ont une idée qui va révolutionner le domaine de la cryptographie. Ils proposent en effet une solution au problème de l'échange de clés et et à la signature électronique : cette solution est la cryptographie à clé publique. Le principe repose sur une paire de clés : une clé publique pk qui est utilisée pour chiffrer ou vérifier une signature, et une clé secrète sk qui sert à déchiffrer ou à signer. La clé publique est, comme son nom l'indique, accessible à tous de telle sorte que tout le monde est capable de chiffrer un message ou de vérifier une signature. La clé secrète est quant à elle non diffusée donnant ainsi uniquement à son propriétaire la capacité de déchiffrer les messages qui ont été chiffrés avec la clé publique associée, ou de produire des signatures qui seront vérifiables avec cette même clé publique.

1.2.1 Schéma de chiffrement à clé publique

Un schéma de chiffrement à clé publique inclut une paire de fonctions : la fonction de chiffrement Enc paramétrée par la clé publique pk et la fonction de déchiffrement Dec paramétrée par la clé secrète sk. Ces deux fonctions sont de plus reliées par la relation $\mathsf{Dec}_{\mathsf{sk}} \circ \mathsf{Enc}_{\mathsf{pk}} = \mathsf{Id}$. Pour assurer la confidentialité d'un message que l'on souhaite envoyer à quelqu'un, on récupère tout d'abord sa clé publique pk. Le chiffrement du message m en un message chiffré c est alors défini par $c = \mathsf{Enc}_{\mathsf{pk}}(m)$. Pour retrouver le message initial, le destinataire du message chiffré doit quant à lui effectuer l'opération $m = \mathsf{Dec}_{\mathsf{sk}}(c)$. Pour qu'un tel schéma soit sûr, il doit être calculatoirement impossible d'évaluer la fonction de déchiffrement Dec sans avoir la clé secrète sk, même en ayant connaissance de la clé publique pk.

Malgré un certain nombre d'avantages indéniables que proposent les schémas à clé publique, ils ont également un inconvénient majeur : ils sont en pratique bien moins efficaces que les schémas symétriques. En conséquence, on utilise souvent de nos jours des *schémas de chiffrement hybride* qui utilisent un schéma à clé publique pour l'échange d'une clé secrète, soit un mot de quelques octets, puis l'utilisation d'un schéma symétrique pour le chiffrement des messages, mots de potentiellement plusieurs milliers d'octets.

1.2.2 Signature électronique

Les signatures électroniques ont le même rôle qu'une signature classique sur papier : elles permettent d'authentifier l'auteur du document signé et de vérifier l'intégrité du message. De plus, contrairement à d'autres méthodes d'authentification offertes par la cryptographie symétrique comme les MAC, une autre propriété très importante, dans le domaine du commerce par exemple, est offerte avec ce type de schéma : la *non-répudiation*, c'est-à-dire l'impossibilité de remettre en cause la signature.

Tout schéma de chiffrement à clé publique peut également être utilisé pour la signature électronique de documents. Les deux fonctions mises en jeux sont ici la fonction de signature Sign et la fonction de vérification Verif respectivement paramétrées par la clé secrète sk et la clé publique pk. Ainsi la signature s d'un message m peut être obtenue en effectuant l'opération $s = \mathsf{Sign}_{\mathsf{sk}}(m)$. Toute personne ayant accès à m, s et la clé publique pk du prétendu signataire peut vérifier que le message est bien de lui grâce au test $\mathsf{Verif}_{\mathsf{pk}}(s) \stackrel{?}{=} m$. La sécurité d'un schéma de signature électronique repose sur le fait qu'il

est calculatoirement impossible d'évaluer, connaissant la clé publique pk, la fonction de signature Sign sans la clé secrète sk.

1.3 Problèmes difficiles

La fonction de chiffrement Enc d'une primitive à clé publique doit être efficacement calculable pour être concrètement utilisable et être très difficile à inverser pour empêcher quiconque n'a pas la bonne clé secrète de retrouver le message m. En effet, le chiffré $c = \mathsf{Enc}_{\mathsf{pk}}(m)$ ne dépend potentiellement (si aucun aléa n'est mis en jeu) que de données considérées comme publiques (excepté m bien sûr) et peut circuler via un canal non sécurisé, donc peut être intercepté. En d'autres mots, on souhaite que cette fonction de chiffrement soit une *fonction à sens unique* mais l'existence de telles fonctions est toujours un problème ouvert. En fait, leur existence prouverait que les classes de complexité P et NP ne sont pas égales, ce qui résoudrait la question ouverte la plus célèbre de l'informatique théorique. Ainsi, la cryptographie à clé publique repose sur des conjectures et les primitives cryptographiques utilisent des problèmes NP supposés difficiles.

1.3.1 Factorisation et RSA

La première[1] réalisation d'un primitive de chiffrement à clé publique est proposée en 1978 par Ron Rivest, Adi Shamir et Leonard Adleman [RSA78], et connue aujourd'hui sous le nom de "textbook RSA". Pour cette primitive, deux grands nombres premiers p et q sont tirés aléatoirement et un *module public* $N = pq$ est défini. Ensuite, on choisit un *exposant public* $e \in \mathbb{Z}$ premier avec $\varphi(N) = (p-1)(q-1)$ et, puisque e est inversible modulo $\varphi(N)$, il existe un entier $d \in \mathbb{Z}_{\varphi(N)}$ tel que $e \cdot d = 1 \bmod \varphi(N)$. La clé publique pk est représentée par la paire (e, N) alors que la clé secrète sk est l'*exposant secret* d. Le chiffrement d'un message $m \in Z_N^*$ s'effectue en calculant :

$$c = m^e \bmod N,$$

alors que le déchiffrement de $c \in Z_N^*$ s'obtient, grâce au théorème d'Euler, par le calcul :

$$m = c^d \bmod N.$$

Le problème consistant à retrouver la clé secrète grâce à la clé publique est essentiellement équivalent au problème de la factorisation de N. En effet, si l'on connaît les deux facteurs de N, à savoir p et q, alors on peut calculer $\varphi(N)$ puis l'inverse de e par cette valeur. Réciproquement, si l'on connaît les deux exposants e et d, alors on peut calculer un multiple de $\varphi(N)$ puis $\varphi(N)$ grâce à l'algorithme de Miller-Rabin. Enfin, à partir de $p + q = N - \varphi(N) + 1$ et $pq = N$, il devient possible de retrouver p et q. Mais factoriser un produit de deux grands nombre premiers, ici N, est une tâche difficile dans le sens où on ne sait pas le faire efficacement : le meilleur algorithme connu, le crible algébrique, est sous-exponentiel en la taille de N. Ainsi, la factorisation est aujourd'hui l'un des problèmes supposés difficiles les plus utilisés en cryptographie.

Une autre possibilité pour cryptanalyser la primitive RSA consiste à retrouver m à partir de c sans utiliser la clé secrète sk, autrement dit à calculer la racine e-ième de c

[1] En réalité, un document de 1969, mais déclassifié seulement dans les années 90, de James Ellis, qui travaillait pour l'agence britannique de renseignement électronique (GCHQ), mentionne déjà le principe de la cryptographie à clé publique. De plus, en 1973, Clifford Cocks, toujours du GCHQ, a développé une primitive cryptographique complètement analogue à la primitive RSA, à savoir $x \to x^N \bmod N$.

modulo N. Ce problème, connu sous le nom de *problème RSA*, est a priori plus facile que le problème de la factorisation puisque la factorisation de N permet de calculer des racines e-ièmes, alors que retrouver la factorisation de N à partir de racines e-ièmes est toujours un problème ouvert. Cependant le meilleur algorithme connu pour calculer des racines e-ièmes est celui utilisé pour la factorisation. Il n'est ainsi pas clair à l'heure actuelle si les deux problèmes sont équivalents (il existe des indices faisant penser que non [BV98]), mais bien qu'il soit fréquent d'entendre que RSA repose sur la difficulté supposée de la factorisation, une preuve de cette difficulté ne signifierait pas que RSA serait sûr.

Enfin, il est important de noter que le "textbook RSA" n'est pas sûr tel qu'il est présenté car il est déterministe et ne possède donc pas une propriété de sécurité importante qui est l'*indistinguabilité* des chiffrés. En effet, on souhaite qu'un même message chiffré à deux moments différents ne donne pas le même résultat afin de ne donner aucune information à une personne qui espionnerait le canal de communication. Plus généralement même, on demande que le résultat d'un chiffrement soit indistinguable d'une valeur totalement aléatoire et qu'aucun bit d'information sur le message ne puisse en être extrait. Il existe aujourd'hui de nombreux schémas de chiffrement et de signature qui sont prouvés sûrs sous l'hypothèse que le problème RSA l'est également et ces schémas sont parmi les plus utilisés dans le monde même si leur hégémonie s'estompe peu à peu.

1.3.2 Logarithme discret et problèmes Diffie-Hellman

Comme précisé précédemment, l'idée de la cryptographie à clé publique est attribuée à Whitfield Diffie et Martin Hellman [DH76] et leur première réalisation est un protocole d'échange de clé sécurisé via un canal non sécurisé. Plus précisemment, en notant q un grand nombre premier et g un générateur d'un grand sous-groupe $\mathbb{G}$ d'ordre premier p de $\mathbb{Z}_q^*$, l'échange se déroule de la façon suivante. Chacun des deux participants tire aléatoirement un élément de $\mathbb{Z}_p$, noté x (respectivement y), puis envoie à son partenaire g^x (respectivement g^y). Ainsi les deux sont maintenant capables de calculer la valeur $g^{xy} = (g^x)^y = (g^y)^x$ qui devient leur clé commune. Un adversaire peut voir passer sur le canal les valeurs g^x et g^y et cassera le protocole s'il est capable, à partir de ces deux valeurs, de calculer g^{xy} : c'est le *problème Diffie-Hellman calculatoire* (CDH). La meilleure attaque consiste en fait à retrouver x connaissant g^x (ou similairement avec y), il s'agit ici du *problème du logarithme discret* dans $\mathbb{Z}_q^*$. Ce dernier problème est solvable par un algorithme de crible algébrique comme pour la factorisation. De nombreux schémas cryptographiques, basés sur le problème du logarithme discret ou le problème CDH voir une autre variante, existent tels que le schéma El Gamal [Gam84] ou le très répandu schéma de signature DSA [FIP09].

En 1985, Neal Koblitz [Kob87] et Victor S. Miller [Mil85] ont indépendamment proposé de porter le problème du logarithme discret sur les courbes elliptiques, c'est-à-dire de considérer $\mathbb{G}$ comme un sous-groupe cyclique du groupe des points d'une courbe elliptique sur un corps fini : c'est la naissance de la *cryptographie par courbes elliptiques*. L'avantage réside principalement dans le fait que le problème du logarithme discret (ou un problème apparenté) a toujours, a priori, une complexité exponentielle sur les courbes elliptiques alors qu'elle est sous-exponentielle dans les corps finis. En conséquence, la taille des clés secrètes peut être significativement réduite, améliorant ainsi l'efficacité des schémas cryptographiques. Cet argument de poids pousse le monde industriel a privilégier la cryptographie par courbes elliptiques, qui devient de plus en plus populaire [BHH+13] et menace ainsi l'hégémonie de schémas basés sur le problème RSA.

1.3.3 Autres problèmes

Les problèmes décrits précédemment sont très majoritairement ceux utilisés concrètement de nos jours, mais il en existe bien d'autres. On peut citer les problèmes de sac à dos, les problèmes de décodage pour les codes linéaires ou encore les problèmes de résolution de grands systèmes polynomiaux. Les problèmes associés aux *réseaux*, c'est-à-dire les sous-groupes discrets de $\mathbb{R}^n$, forment un sujet de recherche assez prolifique. Les deux principaux sont la recherche d'un plus court vecteur non nul et la recherche d'un plus proche vecteur à un point donné de l'espace ambiant. Il existe un certain nombre de constructions basées sur ces problèmes mais, bien qu'elles soient "asymptotiquement efficaces", elles requièrent concrètement des paramètres relativement grands, ce qui limite leur efficacité et leur utilisation.

1.3.4 Sécurité heuristique et sécurité prouvée

Comme expliqué auparavant, la sécurité des primitives ou des schémas à clé publique repose sur des problèmes conjecturés difficiles. Il s'agit maintenant de répondre à la question : quand peut-on dire qu'un schéma cryptographique est (heuristiquement) sûr ? Le "textbook RSA" ne l'est pas, même s'il repose sur le problème de la factorisation, car on peut déterminer si deux chiffrés correspondent au même message et on peut décrypter, c'est-à-dire déchiffrer sans connaître la clé secrète, les chiffrés de messages trop courts (suffisamment courts pour que la réduction modulo N ne se fasse pas).

Afin de prouver qu'un schéma à clé public est sûr, le cryptographe considère un scénario d'attaque dans lequel un adversaire a accès au schéma en *boîte noire*. En d'autres mots, il a à sa disposition les entrées (les messages) et les sorties (les chiffrés, les signatures...) de cette boîte virtuelle contenant les différents algorithmes mis en jeu, algorithmes qu'il connaît également selon le principe de Kerckhoffs. La clé secrète et les résultats intermédiaires restent par contre inaccessibles. L'objectif du cryptographe est alors de démontrer qu'un tel adversaire ne peut rien tirer des informations qu'il possède. Il effectue pour cela une *preuve de sécurité par réduction* en modifiant au fur et à mesure les calculs effectués en boîte noire, en prouvant que ces modifications ne peuvent pas être perçues par l'adversaire (les sorties de la boîte noire lui semblent provenir du schéma initial), et ce jusqu'à faire apparaître clairement un problème conjecturé difficile. Ainsi, si un adversaire est capable d'attaquer efficacement, comprendre en temps polynomial, le schéma visé alors il peut résoudre efficacement le problème conjecturé difficile. Cependant, le problème étant conjecturé justement difficile, il ne peut être résolu efficacement et donc un tel adversaire ne peut exister.

1.4 Implémentation et cryptanalyse physique

1.4.1 Implémentation et problèmes

Le modèle de la boîte noire est très pratique pour caractériser la sécurité intrinsèque d'un schéma cryptographique mais il ne permet pas de prouver sa sécurité dans le monde physique, la description mathématique du schéma étant remplacée par une implémentation physique pour que des dispositifs, tels que les ordinateurs ou les cartes à puce, puissent effectuer les calculs cryptographiques. Le problème majeur avec une implémentation physique exécutant ce genre de calculs est qu'elle laisse apparaître des informations physiques observables sur les résultats intermédiaires voir la clé secrète. On peut citer le temps d'exécution, la consommation électrique et les radiations électromagnétiques comme exemples

d'information qui peut fuir du dispositif [Koc96, KJJ99]. De plus, une implémentation physique n'est pas inviolable dans le sens où elle peut être altérée ou bien certains calculs peuvent être perturbés. Ceci est réalisable grâce à une brève très forte augmentation de tension ou avec l'aide d'un laser par exemple. Bien entendu, l'observation d'informations physiques, sauf potentiellement le temps d'exécution, et la perturbation de calculs demandent d'avoir un accès physique à l'implémentation. Cet accès est en pratique possible pour les implémentations cryptographiques embarquées dans des produits comme les cartes bancaires ou les cartes d'accès TV par exemple.

1.4.2 Cryptanalyse physique

Des attaques d'un genre différent ont ainsi émergé dans ce contexte et sont regroupées sous le nom de *cryptanalyse physique*. Plus précisément, on peut distinguer deux catégories d'attaques physiques qui utilisent des méthodes distinctes : les *attaques par canaux cachés* et les *attaques par faute*. Les attaques par canaux cachés exploitent les fuites d'informations physiques qui apparaissent lors de calculs cryptographiques. Ces fuites sont collectées et analysées pour retrouver la clé secrète du schéma cryptographique qui peut, par ailleurs, être prouvé sur dans le modèle de la boîte noire. Les attaques par faute sont quant à elles invasives puisqu'elles consistent à gêner un calcul cryptographique, au travers d'injection d'une ou plusieurs fautes. Le résultat de ce calcul est par conséquent incorrect mais son analyse peut permettre de retrouver de l'information sur la clé secrète du schéma. Ce type de cryptanalyse et le développement de *contre-mesures* pour s'en prémunir est un domaine de recherche très actif de nos jours.

1.5 Générateurs d'aléas

Clés secrètes, modes opératoires, contre-mesures sont des exemples d'objets cryptographiques nécessitant de générer des suites de bits aléatoires. Il n'est cependant que rarement détaillé comment obtenir de telles suites et les preuves de sécurité conjecturent souvent qu'elles sont "de bonne qualité". Or si la méthode de génération utilisée n'est pas bonne, elle peut constituer une véritable faille de sécurité pour le schéma cryptographique concerné [HDWH12, LHA+12, BCC+13]. Générer des suites de bits aléatoires est en réalité une vraie difficulté car même si le hasard existe bien, comment en engendrer ? Comment le mesurer ? On peut distinguer trois catégories de générateurs de nombres aléatoires : les *générateurs vraiment aléatoires* ou TRNG, les *générateurs pseudo-aléatoires* ou PRNG, et les *générateurs hybrides* ou HRNG.

1.5.1 Générateur vraiment aléatoire

Un générateur composé d'une *source de bruit* et d'un traitement simple qui corrige les défauts ou *biais* de cette source est appelé générateur vraiment aléatoire. La source de bruit correspond à la partie non-déterministe du générateur et provient de phénomènes physiques dont le comportement est au moins partiellement aléatoire. On peut citer la radioactivité, les bruits thermiques ou électromagnétiques et la mécanique quantique à titre d'exemple. La sortie d'une source de bruit est une chaîne binaire. Le traitement, déterministe, prend en entrée cette chaîne de bits potentiellement biaisée et renvoie une suite binaire totalement imprédictible. Il doit prendre en compte le fait que la source de bruit peut ne pas fonctionner comme prévue, par exemple si le phénomène physique est perturbé par d'hypothétiques attaques. Pour qu'un tel générateur soit sûr, on demande que la chaîne binaire finale ait de bonnes propriétés statistiques et qu'elle soit imprédictible

(voir [BLMT11] pour un exemple d'analyse de sécurité). L'inconvénient de ce type de générateur est qu'il est relativement lent.

1.5.2 Générateur pseudo-aléatoire

Un générateur pseudo-aléatoire est un algorithme déterministe qui prend en entrée une graine aléatoire de k bits et renvoie une suite de K bits, avec $K \gg k$. Cette suite est appelée suite pseudo-aléatoire et doit être difficilement distinguable d'une suite de bits aléatoires.

Ces générateurs ont généralement de bonnes propriétés statistiques mais il faut s'assurer, dans le cas d'une utilisation en cryptographie, d'une part que leur période est vraiment grande sous peine d'avoir de la redondance, d'autre part qu'il est calculatoirement sûr, c'est-à-dire qu'il doit être calculatoirement difficile de prédire, à partir d'une sous-suite de la suite pseudo-aléatoire, l'élément précédent ou suivant de celle-ci. Par ailleurs, une autre propriété est souhaitée pour parler de *générateur pseudo-aléatoire cryptographiquement sûr* : connaissant un élément de la suite et l'état interne du générateur au moment du calcul de cet élément, on est logiquement capable de prédire les éléments suivants mais il doit être calculatoirement difficile de remonter aux éléments précédents.

De nombreuses constructions de générateurs pseudo-aléatoires cryptographiquement sûrs ont été proposés comme celle de Blum et Micali [BM84], celle de Blum, Blum et Shub [BBS86] ou celle de Micali et Schnorr [MS91]. Ces générateurs reposent sur des problèmes supposés difficiles et sont disposent d'une preuve de sécurité par réduction. Ils sont toutefois très lents et ne sont utilisés en pratique que quand cette preuve de sécurité de la suite pseudo-aléatoire est nécessaire.

1.5.3 Générateur hybride

La dernière catégorie de générateurs reprend le principe de fonctionnement du générateur pseudo-aléatoire avec, en prime, introduction régulière d'aléa dans l'état interne pour le rafraîchir afin d'empêcher que le générateur ne devienne prévisible si cet état est compromis, à savoir connu ou modifié. L'aléa injecté provient d'un générateur vraiment aléatoire et on appelle ces générateurs des générateurs hybrides. Ils sont en général construits à partir de primitives cryptographiques fortes, dont la sécurité n'est pas prouvée mais admise, comme la fonction de hachage SHA-256 ou l'algorithme de chiffrement par bloc AES, afin de gagner en rapidité.

Bien entendu, cette introduction ne présente que quelques éléments de la cryptologie avec un certain nombre de notions et principes. Ainsi, la cryptographie à clé publique a largement été favorisée car elle est au cœur de ce manuscrit de thèse tout comme les générateurs pseudo-aléatoires. Le livre [MVO96] est une excellente référence pour compléter ce chapitre.

Chapitre 2
Présentation des travaux

Les travaux présentés dans ce manuscrit ont principalement porté sur deux thèmes distincts de la cryptologie, à savoir les générateurs pseudo-aléatoires et les schémas à clé publique, essentiellement de signature. Ainsi, d'un côté nous avons étudié « l'aléa » généré par différents générateurs pseudo-aléatoires algébriques, le terme aléa étant mis entre guillemets car une majorité de nos travaux sont des cryptanalyses algébriques exploitant sa « mauvaise qualité ». D'un autre côté nous nous sommes beaucoup intéressés aux schémas de signature à clé publique, essentiellement les schémas RSA et ECDSA. Plus précisément, nous avons analysé la sécurité d'implémentations pratiques de ces schémas et l'impact d'une injection de fautes sur leur sécurité. Il aurait du coup été naturel de fractionner ce manuscrit en deux parties mais les travaux sur RSA et sur courbe elliptique ont représenté le plus gros des recherches, d'où un découpage en trois parties : la première porte sur l'impact des fautes sur des implémentations du schéma de signature RSA, la seconde détaille nos travaux sur courbe elliptique et plus spécifiquement des implémentations, la dernière partie est quant à elle consacrée à l'étude de générateurs pseudo-aléatoires.

Concernant le schéma de signature RSA, nous avons concentré nos recherches sur RSA–CRT qui permet de calculer bien plus rapidement des signatures. Nous avons proposé, dans le cas d'une implémentation très utilisée en pratique, différentes attaques par injection de faute qui permettent de retrouver la clé secrète quelque soit l'encodage du message utilisé. L'idée de descendre de plus en plus bas dans les détails d'implémentation pour trouver des failles exploitables nous a conduit à élaborer un outil qui recherche automatiquement des attaques par faute à partir d'une implémentation précise et des modèles de faute que nous voulions considérer. Enfin, nous avons proposé une contre-mesure infective pour protéger le schéma RSA–PSS contre un certain nombre de fautes non aléatoires, le schéma ainsi modifié ayant été prouvé formellement grâce à un outil de preuve assistée par ordinateur.

La seconde partie porte sur des implémentations de deux schémas sur courbe elliptique : le schéma de signature ECDSA et un schéma nommé Elligator Squared qui participe à la préservation de l'anonymat et la vie privée. Des implémentations très efficaces pour accélérer les calculs de signatures ECDSA utilisent la décomposition GLV/GLS qui cible le nonce utilisé dans le schéma. Nous avons montré que, en fonction de la méthode utilisée pour générer ce nonce, nous pouvions soit prouver la bonne distribution de celui-ci soit démontrer qu'il était biaisé sur un bit. Pour un module de 160 bits, ce biais a été exploité pour retrouver la clé secrète, ce qui représente un record, les attaques existantes nécessitant un biais plus élevé pour réussir. Nous avons également proposé une attaque par canaux cachés pour une implémentation particulière et soumis des implémentations de ECDSA à notre outil de recherche automatique d'attaques par faute, outil qui a trouvé plus de cent combinaisons différentes de fautes sur une implémentation très bas niveau. Quant au schéma Elligator Squared, il permet de transformer un point d'une courbe elliptique en une chaîne de bits indistinguable d'une chaîne de bits aléatoire, ce qui permet de masquer l'utilisation de protocoles cryptographiques à base de courbe elliptique sur un réseau. Nous avons implémenté ce schéma pour la courbe Curve25519 afin de comparer son efficacité par

rapport à un schéma similaire, et nous avons effectué un travail d'implémentation efficace dans le cas d'une courbe sur un corps fini de caractéristique deux afin de démontrer que le surcoût d'une telle protection est relativement minimal.

Enfin, la dernière partie est dédiée à la sécurité d'un certain nombre de générateurs pseudo-aléatoires algébriques. Nous avons ainsi étudié tous les générateurs dont la fonction de récurrence de l'état interne est un polynôme quelconque et proposé de meilleures bornes d'attaque par rapport à la littérature, c'est-à-dire des attaques effectives quand moins de bits sont générés par itération de ces générateurs par rapport à la quantité nécessaire auparavant. En outre, nous avons étudié la sécurité du générateur Micali-Schnorr, en analysant ses faiblesses et les propriétés statistiques des bits générés. Finalement, nous avons proposé une cryptanalyse de tout schéma à clé publique basé sur la factorisation ou le logarithme discret dont la clé secrète est générée à partir d'un générateur congruentiel linéaire.

2.1 Fautes sur des implémentations du schéma de signature RSA

Cette première partie est donc consacrée aux implémentations de schémas de signature RSA et notamment sur l'impact de fautes sur leur sécurité. Les premiers travaux présentent des attaques par faute, trouvées manuellement ou automatiquement alors que la dernière contribution propose une preuve de sécurité contre certains modèles d'attaques par faute.

2.1.1 Attaquer des signatures RSA–CRT avec des fautes sur la multiplication de Montgomery [FGL+12, FGL+13]

Une des implémentations les plus efficaces pour calculer une signature à l'aide de l'algorithme RSA consiste à combiner deux techniques. La première est connue sous le nom de RSA–CRT et consiste à calculer la signature modulo p et modulo q, puis combiner les deux résultats à l'aide du théorème des restes chinois (CRT) ou d'une variante, la formule de Garner. Comparé à une classique exponentiation modulo $N = pq$, de complexité cubique en la taille du module, le gain en temps de calcul est de l'ordre d'un facteur 4. La seconde technique cible plus spécifiquement les calculs de multiplication modulaire qui sont le cœur d'une exponentiation modulaire : il s'agit de la multiplication de Montgomery. Son intérêt réside dans le fait qu'elle remplace les coûteuses divisions sur grands entiers par des multiplications non modulaires et des décalages. Ainsi une multiplication modulaire calculée avec cette technique coûte l'équivalent de deux multiplications non modulaires.

Il faut cependant noter que les implémentations basiques de RSA–CRT sont très vulnérables aux attaques par faute. Ce fait a été initialement démontré en 1997 par Boneh, DeMillo et Lipton et leur résultat a ouvert la voie à une nouvelle thématique de recherche très prolifique : trouver d'une part des attaques par faute sur les signatures RSA et proposer d'autre part des contre-mesures à ces attaques. L'attaque initiale, relativement simple et très puissante, consiste à injecter une faute aléatoire pendant le calcul d'une signature modulo p ou modulo q. La signature fautée permet alors de retrouver la factorisation du module RSA grâce à un calcul de PGCD. Cependant, ce calcul n'est possible que si le message est connu, ce qui est vrai si aucun encodage n'a été effectué sur le message en entrée ou s'il est déterministe. Ainsi, si l'encodage est probabiliste, cette attaque n'est pas directement applicable même si des généralisations de celle-ci ont été publiés dans certains cas comme les signatures ISO/IEC 9796-2 ou les signatures EMV. C'est dans ce contexte que Coron et Mandal ont prouvé que le schéma d'encodage de Bellare et Rogaway RSA–PSS est sûr contre les fautes aléatoires dans le modèle de l'oracle aléatoire.

Il pourrait être tentant de conclure que l'utilisation de cet encodage dispense d'implémenter d'autres coûteuses contre-mesures supplémentaires. Nous avons démontré que cela n'est pas vrai en nous plaçant dans le cas d'une implémentation utilisant la multiplication de Montgomery. Plus précisément, nous avons proposé de nouvelles attaques par faute applicables quelque soit la technique de recombinaison utilisée, à savoir le théorème des restes chinois ou la formule de Garner, et quelque soit l'encodage utilisé, probabiliste ou non. Ces nouvelles attaques sont ainsi les premières à être efficaces contre RSA–PSS.

Nos attaques sont classées selon trois différents modèles de faute ciblant un petit registre utilisé pendant les multiplications de Montgomery, typiquement de 16, 32 ou 64 bits. Dans le premier modèle, qui est également le plus efficace, nous forçons la valeur contenue dans un petit registre précis à zéro. La signature fautée résultante s'avère être alors un multiple de p ou de q et un simple calcul de PGCD conclut l'attaque. Le deuxième modèle consiste à forcer la valeur contenue dans un autre petit registre à une constante, potentiellement inconnue, pendant un certain laps de temps. La signature fautée a dans ce cas la propriété d'être un proche multiple de p ou de q et un algorithme de commun diviseur approché permet de retrouver la factorisation de N. Enfin, dans un troisième modèle, nous forçons un certain nombre de bits de poids fort d'une valeur contenue dans un petit registre à zéro. Le résultat n'est alors pas assez proche d'un multiple de p ou de q mais l'obtention d'un nombre modéré de signatures fautées selon ce dernier modèle permet tout de même de finir l'attaque. Toutes nos attaques diffèrent selon la méthode de recombinaison utilisée, les plus simples et efficaces étant dans le cas de la formule de Garner.

Enfin, nous avons discuté du réalisme de nos modèles de faute en analysant différentes implémentations matérielles de la multiplication de Montgomery et nous démontrons, en pointant les zones vulnérables, qu'une majorité de ces implémentations sont sensibles à nos attaques.

2.1.2 Recherche automatique d'attaques par faute sur des implémentations cryptographiques [BDF+14b]

Une des leçons que nous avons tirées de nos résultats est que la recherche manuelle d'attaque par faute sur une implémentation est une tâche relativement fastidieuse, surtout quand le code de l'implémentation en question s'étale sur des dizaines de lignes (si on explicite chaque multiplication modulaire par exemple) et qu'il met en jeu un grand nombre de variables temporaires. Nous avons donc voulu automatiser cette recherche mais de manière intelligente : une méthode naïve et extrêmement coûteuse consisterait à produire tous les algorithmes fautés possibles selon le(s) modèle(s) de faute considéré(s) et de vérifier si les signatures résultantes permettent de retrouver la clé secrète. Pour avoir un outil efficace et performant, nous avons effectué des travaux dans deux directions.

Premièrement, nous avons identifié des *conditions de faute*, un nouveau concept, qui peuvent être vues comme des propriétés mathématiques indépendantes de toute implémentation, spécifiques à un schéma cryptographique et capturant suffisamment de conditions nécessaires pour réussir une attaque, c'est-à-dire retrouver la clé secrète. Par exemple, un attaquant ayant à sa disposition, dans le cadre de RSA, une valeur S qui est un multiple de p mais pas de q peut retrouver la valeur de p grâce à un simple PGCD. Ceci est capturé par la condition de faute :

$$S : S = 0 \bmod p \wedge S \neq 0 \bmod q.$$

Nous avons également considéré les deux conditions de faute suivantes dans le cadre de nos

travaux :

$$\exists \alpha, \beta \quad S = \alpha\ p + \beta\ q \wedge \alpha, \beta < 2^{\frac{n}{2}-\varepsilon},$$
$$\exists \alpha, \beta \quad S = \alpha\ p + \beta \wedge \alpha < q, \beta < 2^{n/2-\varepsilon}.$$

Deuxièmement, nous avons proposé une méthode automatisée pour découvrir des implémentations fautées vérifiant la condition de faute souhaitée. Cette méthode peut être perçue comme une instance de la synthèse de programme, dont l'objectif général est de trouver, pour une spécification ϕ donnée, un ensemble de programmes satisfaisant ϕ. Plus spécifiquement, notre algorithme prend en entrées une condition de faute ϕ, une implémentation c, et cherche toutes les implémentations fautées de c satisfaisant ϕ. En outre, la recherche est restreinte grâce à deux entrées supplémentaires : une *politique de faute* qui décrit précisément quelles fautes peuvent être injectées dans le programme (plusieurs modèles de fautes peuvent être englobés une politique de faute), et une borne supérieure sur le nombre de fautes injectables.

Enfin, nous avons évalué notre approche sur l'implémentation de RSA–CRT avec multiplication de Montgomery. Nous avons ainsi retrouvé l'attaque utilisant le premier modèle de faute, des variantes de celle-ci, et de nouvelles attaques. La politique de faute utilisée autorisait notre premier modèle de faute ainsi que les fautes sur le flot de contrôle. Plus précisément, nous autorisions des fautes qui retiraient ou ajoutaient une ou plusieurs itérations d'une boucle. Certaines des nouvelles attaques que nous avons trouvées sont particulièrement efficaces dans le cas de la recombinaison par le théorème des restes chinois.

2.1.3 Protéger RSA–PSS de façon prouvée contre des fautes non aléatoires [BDF+14a]

Après avoir analysé le schéma RSA–CRT d'un point de vue offensif, nous avons regardé comment le protéger. Plus précisément, nous avons considéré le schéma de signature RSA–PSS utilisant le théorème des restes chinois pour accéler le calcul de l'exponentiation modulaire et nous avons proposé une contre-mesure infective. Ce type de contre-mesure a l'avantage de pouvoir résister à une attaque qui, en plus de fausser le calcul, tromperait le résultat d'un test d'intégrité dont le but est de détecter une erreur de calcul : si celui-ci est correct alors la signature est donnée en résultat, sinon un message d'erreur est retourné. En effet, si on suppose que l'attaquant a la possibilité de passer outre les tests d'intégrité, alors des attaques comme celles que nous avons présentées réussiront même si des calculs de vérification intermédiaires ont été effectués. Les contre-mesures infectives *infectent* le résultat de telle façon que si le calcul s'est déroulé correctement, alors la signature sera correcte, sinon le résultat sera a priori inutilisable par l'attaquant.

Nous avons ainsi considéré le modèle d'attaque suivant : l'attaquant peut obtenir une signature fautée de la forme $(y^d \bmod p, a) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}$, avec a une valeur de son choix. En d'autres termes, nous l'autorisons à fauter le calcul pendant une des deux demi-exponentiations et à fixer son résultat à une valeur précise. Ce modèle d'attaque englobe donc les fautes aléatoires qui étaient prises en compte par Coron et Mandal, mais également le résultat de notre attaque utilisant le premier modèle de faute, c'est-à-dire que la signature modulo p ou q vaut zéro, en posant $a = 0$.

Sous ce modèle d'attaque, nous avons démontré que la sécurité de ce schéma protégé de RSA–PSS se réduisait à la sécurité du problème RSA. En outre, notre preuve a été formellement vérifiée grâce à l'utilisation d'un outil de preuve assistée par ordinateur nommé EasyCrypt. Cet outil a par ailleurs révélé un léger problème dans la preuve initiale, ce qui

justifie encore davantage la nécessité croissante de vérifier formellement les preuves par des outils tels que celui-là.

2.2 Sécurité d'implémentations du schéma ECDSA, Efficacité du schéma Elligator Squared

La seconde partie de ce manuscrit détaille mes travaux sur courbe elliptique. Certains ont porté sur l'algorithme de signature ECDSA alors que d'autres ont consisté à implémenter efficacement un schéma qui masque l'échange des coordonnées d'un point d'une courbe elliptique sur un réseau, Elligator Squared.

2.2.1 Décomposition GLV/GLS et sécurité d'implémentations du schéma ECDSA [BDF+14b, AFG+14]

Lors du calcul d'une signature ECDSA, la multiplication scalaire $[k]P$, avec P un point d'ordre premier n de la courbe elliptique utilisée et k un nonce de taille pleine, est l'étape la plus coûteuse. C'est à partir de ce constat que sont apparues les techniques GLV de Gallant, Lambert, Vanstone et GLS de Galbraith, Lin, Scott. Celles-ci sont assez similaires à l'astuce RSA–CRT dans le sens où la multiplication scalaire avec le nonce k de taille pleine est transformée en deux multiplications scalaires avec des nonces k_1, k_2 dont la taille est de l'ordre de $\sqrt{n}$. Plus précisément, sur des courbes elliptiques dotées d'un endomorphisme rapide et non trivial, le résultat du calcul $[k]P$ peut se retrouver par le calcul $(k_1 + k_2\lambda)P$ avec λ une constante donnée. Deux possibilités sont offertes : générer aléatoirement k_1 et k_2 puis calculer k, c'est la technique de recomposition, ou bien décomposer k en k_1 et k_2, c'est la technique de décomposition.

En se plaçant sur des courbes obtenues par la méthode GLS, nous avons prouvé que si les valeurs de k_1 et k_2 sont uniformément distribuées sur l'intervalle $[0, \sqrt{n})$, alors la distribution de valeur de k obtenue par la technique de recomposition est statistiquement proche de la distribution uniforme, et donc les protocoles résultants sont sûrs. Par contre, si les valeurs de k_1 et k_2 sont choisies de la même façon mais sur un intervalle plus restreint, à savoir $[0, 2^m)$ avec $m = \lfloor \frac{1}{2} \log_2 n \rfloor$, alors nous avons montré que la valeur de k est légèrement biaisée, de l'ordre de un bit.

Les biais sur le nonce k ont été la source de nombreuses publications permettant de retrouver la clé secrète. L'idée consiste à exprimer le problème de reconstruction de la clé secrète en une instance du problème du nombre caché (HNP). Les plus puissantes attaques en terme de complexité et de quantité de signatures nécessaire sont basées sur la réduction de réseau car le problème HNP se réduit au problème du plus proche vecteur (CVP). Ainsi Liu et Nguyen ont proposé une attaque contre le schéma ECDSA pour un ordre de 160 bits quand le biais sur k est de deux bits. D'autres attaques existent pour attaquer des courbes ayant un ordre plus grand mais le biais doit être significativement plus élevé. Cependant, pour un biais de un bit, voir moins, l'utilisation des réseaux est impossible, le biais minimum devant être de deux bits.

Bleichenbacher a proposé une attaque, jamais publiée, pour retrouver la clé secrète avec un biais de un bit ou moins en exploitant le biais d'une manière particulière : réduire la taille des signatures par combinaisons linéaires puis retrouver les bits de poids fort de la clé grâce à la Transformée de Fourier Discrète dont les valeurs vont dépendre des biais. L'inconvénient de cette attaque est sa complexité : pour un biais de moins d'un bit et une courbe elliptique ayant un ordre de 160 bits, il faut environ 2^{40} signatures, 2^{47} en temps de calcul et 2^{41} en mémoire. À cette époque, il n'était pas possible de monter l'attaque et seules

des simulations sur des valeurs réduites ont été réalisées. Nous avons, pour la première fois, monté une attaque pour retrouver la totalité de la clé secrète d'un schéma ECDSA avec un biais de un bit. Cette attaque a été réalisée sur une courbe elliptique standardisée (SEC P160 R1) et sur une courbe obtenue par la méthode GLS. Dans le premier cas nos travaux ont consisté à proposer des optimisations algorithmiques pour la phase de réduction des signatures de l'attaque de Bleichenbacher ainsi qu'une implémentation efficace de l'attaque complète, dans le second cas nous avons dû également modifier le biais à exploiter.

En outre, dans le cas d'une implémentation particulière de la technique de décomposition, décrite par Parker et al., nous avons décrit une attaque par canaux cachés utilisant l'information qui fuit lors du calcul d'une multiplication. Cette information permet de retrouver un certain nombre de bits de poids faible du nonce k. Appliquée sur une petite quantité de signatures, la clé secrète peut être retrouvée grâce à la réduction de réseau, comme décrit précédemment. Ce type d'attaque peut se généraliser à d'autres implémentations de la technique de décomposition à partir du moment où le nonce k est manipulé avec des données connues par l'attaquant.

Enfin, nous avons appliqué notre outil de recherche automatique d'attaques par faute sur une implémentation de ECDSA utilisant une implémentation particulière de la multiplication scalaire et nous avons ainsi trouvé de nombreuses nouvelles attaques. Celles-ci ont toutes pour résultat la connaissance d'un certain nombre de bits de poids faible ou de poids fort de k, ouvrant la voie aux attaques par réduction de réseau. L'application du troisième modèle de faute présenté auparavant, des sauts d'instructions ou des fautes sur une instruction de branchement conditionnel permettent d'arriver à ce résultat. En outre, si on raffine l'analyse en prenant en compte une implémentation précise pour l'addition de points, notre outil a trouvé plus de cent programmes fautés, c'est-à-dire combinaisons de fautes permettant de retrouver des bits de k et *in fine* la clé secrète.

2.2.2 Elligator Squared en caractéristique 2 [AFQ+14]

Ce chapitre diffère des précédents dans le sens où il ne traite pas directement d'un schéma de signature ni de sécurité, mais plutôt d'une implémentation efficace d'un schéma, nommé Elligator Squared, pouvant être utilisé pour préserver d'une certaine façon l'anonymat et la vie privée. Le constat est le suivant : un point d'une courbe elliptique, sous forme d'une chaîne de bits, est facilement distinguable d'une chaîne de bits aléatoire. Ainsi, une analyse du trafic réseau d'un protocole implémentant de la cryptographie à base de courbe elliptique peut être aisément discriminé négativement (intercepté, bloqué, trafiqué...).

Bernstein et al. ont proposé en 2013 une approche efficace, appelée Elligator, pour parer à ce problème dans le cas des protocoles cryptographiques basés sur courbe elliptique. Cette approche s'appuie sur l'utilisation d'un encodage injectif ι vers la courbe elliptique, c'est-à-dire qu'un point P de la courbe est représenté par une chaîne de bits $\iota^{-1}(P)$. Les inconvénients de cette approche sont que ces encodages n'existent que pour certaines familles de courbes et qu'ils n'atteignent que la moitié de tous les points possibles pour ces courbes. En 2014 Tibouchi a proposé une variante appelée Elligator Squared pour éliminer ces limitations. Au lieu d'utiliser un encodage injectif, il utilise un encodage ayant de bonnes propriétés statistiques, de la forme $f^{\otimes 2} : (u, v) \mapsto f(u) + f(v)$ avec f un encodage algébrique. Ces encodages existent pour toutes les courbes elliptiques et l'encodage résultant $f^{\otimes 2}$ est essentiellement surjectif, atteignant ainsi tous les points.

La comparaison entre les deux approches n'était jusqu'à présent pas claire en terme de performance concrète. Nous avons ainsi comparé ces deux approches sur une courbe supportée par Elligator (Curve25519) et donné les conclusions suivantes. Pour un proto-

cole utilisant un point de base fixe, la génération d'un point aléatoire et le calcul de sa représentation en chaîne de bits uniforme est environ 35 à 40% plus rapide avec Elligator. Au contraire, pour des protocoles qui utilisent un point de base variable, Elligator Squared est environ 30 à 35% plus rapide.

Par ailleurs, le cœur de nos travaux sur ce sujet a consisté à démontrer que Elligator Squared peut être implémenté très efficacement sur une courbe elliptique de caractéristique 2. En effet, grâce à une version optimisée de l'encodage en caractéristique 2 de Shallue-van de Woestijne et l'utilisation des coordonnées lambda, nous avons pu limiter l'utilisation de la coûteuse opération que représente l'inversion, et utiliser principalement des multiplications, des calculs de trace et de demi-trace. Ainsi notre implémentation efficace de Elligator Squared sur la courbe binaire de Oliveira et al. à CHES 2013 tourne en moyenne en 22850 cycles Haswell. Nous sommes, par ce résultat, bien plus rapide que l'implémentation optimisée de Elligator sur les courbes d'Edward et démontrons qu'il est possible de masquer la transmission de points pour un surcoût minime.

2.3 Sécurité de générateurs pseudo-aléatoires

La dernière partie de ce manuscrit est consacrée à la sécurité des générateurs pseudo-aléatoires non linéaires, du générateur Micali-Schnorr et la sécurité d'un schéma à clé publique utilisant le générateur congruentiel linéaire pour générer sa clé secrète. Tous les travaux suivants consistent en des cryptanalyses, le générateur Micali-Schnorr ayant fait l'objet en outre d'une étude de ses propriétés statistiques.

2.3.1 Retrouver les clés secrètes générées avec des générateurs pseudo-aléatoires faibles [FTZ13]

Le générateur pseudo-aléatoire le plus simple auquel on peut penser est le générateur congruentiel linéaire dont l'état interne évolue selon la récurrence $v_{i+1} = F(v_i) \bmod N$ avec F une fonction affine, N un entier et v_0 la graine secrète, et qui, à chaque itération, révèle une certaine quantité de bits de poids faible (ou de poids fort) de cet état interne. Ce générateur est efficace, facile à implémenter, nécessite peu de mémoire et possède, pour de bons paramètres, de bonnes propriétés statistiques. Il est donc tentant de l'utiliser comme source d'aléa. Cependant, il n'est cryptographiquement pas sûr : avec une séquence plus ou moins longue de sorties, la longueur dépendant du nombre de bits révélé à chaque itération, il a été démontré que l'on peut retrouver la graine secrète en temps polynomial en utilisant la réduction de réseau.

La question que nous nous sommes posés est la suivante : le générateur linéaire est faible et non-sûr quand on a accès à des sorties consécutives de celui-ci. Qu'en est-il s'il est utilisé en boîte noire dans un schéma à clé publique pour générer la clé secrète ? Ce genre de question avait déjà été posé par Bellare, Goldwasser et Micciancio dans le cas de la génération du nonce k pour le schéma de signature DSA et ils avaient démontré l'insécurité d'une telle implémentation. Nous avons regardé un cas plus général dans le sens où nous avons considéré n'importe quel schéma à clé publique basé sur la factorisation ou le logarithme discret, c'est-à-dire les deux problèmes les plus utilisés en cryptographie. Nous montrons que, connaissant la clé publique et les paramètres du générateur linéaire, si la clé secrète est générée par concaténation des sorties de celui-ci alors nous pouvons dans une majorité des cas retrouver cette clé secrète plus rapidement qu'avec une recherche exhaustive sur la graine secrète.

La principale observation de notre travail est la suivante : si on coupe la graine secrète

de k bits en deux morceaux $(A \cdot 2^{k/2} + B)$, la linéarité de ce générateur implique qu'il est possible d'écrire, en oubliant les retenues qui peuvent être gérées indépendamment, la clé secrète comme une somme $U + V \cdot 2^{k/2}$ avec U et V dépendant respectivement uniquement de B et A. Ainsi, on peut effectuer une recherche sur la graine grâce à un compromis temps-mémoire. Le cas du logarithme discret s'apparente à une attaque pas de bébé / pas de géant alors que dans le cas de la factorisation l'attaque utilise l'évaluation polynomiale multi-points.

2.3.2 Exploiter des séquences produites par des générateurs pseudo-aléatoires non linéaires par les méthodes de Coppersmith [BVZ12]

Une suite naturelle des résultats précédents est de considérer des générateurs plus compliqués tels que les générateurs pseudo-aléatoires non linéaires, c'est-à-dire avec F un polynôme de degré $d > 1$. De nombreux travaux ont été effectués pour analyser ces générateurs ou des cas particuliers tels que le générateur quadratique $(F(x) = ax^2 + b)$ et le générateur Pollard $(F(x) = x^2 + b)$. Leurs résultats démontrent globalement qu'il ne faut pas révéler trop de bits par itération sous peine de pouvoir retrouver la graine secrète. Plus précisément, des bornes ont été déterminées telles que si le générateur ciblé révèle plus de bits que cette borne à chaque itération, alors il peut être prédit, étant donné un certain nombre d'itérations. Ces résultats ne prouvent par contre absolument pas que si moins de bits sont révélés, alors le générateur est sûr.

Nos travaux ont consisté à utiliser les techniques de Coppersmith pour tenter d'améliorer les bornes existantes. Coppersmith a présenté en 1996 une technique pour retrouver des petites racines d'un polynôme modulaire univarié et une seconde pour retrouver des petites racines d'un polynôme bivarié sur les entiers. Ces techniques, basées sur la réduction de réseau, ayant eu un fort impact en cryptanalyse algébrique, de nombreuses reformulations et simplifications ont été effectuées. En outre des discussions sur la généralisation de ces techniques aux cas multivariés ont également été proposés car attaquer algébriquement un schéma cryptographique nécessite souvent la manipulation de polynômes ayant un nombre de variables relativement élevé. Dans notre cas, nous avons utilisé la technique de Coppersmith pour des polynômes modulaires multivariés. Sans rentrer dans les détails, l'étape la plus importante et la plus difficile de cette technique consiste à trouver une bonne collection de polynômes avec certaines propriétés algébriques à partir des polynômes du problème initial. Ce choix détermine totalement la taille maximale des racines pouvant être retrouvées, et *in fine* les bornes d'attaque pour les générateurs que nous avons étudiés.

Ainsi, nous avons amélioré toutes les bornes existantes pour un polynôme quelconque de degré $d > 1$, puis, par application plus ou moins direct, les bornes pour le générateur quadratique, et enfin, en utilisant une astuce supplémentaire, les bornes pour le générateur Pollard. Plus précisément, nous avons travaillé sur douze cas différents :

- F quelconque de degré d **ou** $F(x) = ax^2+b$ (générateur quadratique) **ou** $F(x) = x^2+b$ (générateur Pollard),
- F est connu **ou** inconnu (le module est toujours considéré comme connu),
- très peu d'itérations consécutives sont connues **ou** une infinité sont à notre disposition.

Pour chacun de ces cas, nous avons amélioré plus ou moins significativement les bornes. À titre d'exemple, nous avons montré que si un générateur utilisant un polynôme de degré

d pour sa récurrence révèle plus de $d/(d+1)$ des bits de poids fort sur deux itérations consécutives, on peut retrouver la graine secrète, alors que la borne précédente était $(d^2-1)/d^2$. Également, nous avons prouvé qu'asymptotiquement, révéler plus de $2/3$ des bits de poids fort pour un générateur quadratique aux paramètres inconnus est suffisant pour mener une attaque, alors que la borne précédente était $11/12$.

2.3.3 Sécurité du générateur Micali-Schnorr [FVZ13, FZ14]

Le dernier chapitre de cette partie est dédié à l'étude du générateur pseudo-aléatoire Micali-Schnorr qui diffère au niveau de sa récurrence par rapport aux autres générateurs algébriques étudiés jusqu'à présent. En effet, pour ce générateur seul les bits n'ayant pas été révélés sont utilisés pour calculer l'état interne suivant. En d'autres mots, en notant (e, N) la clé publique RSA avec e petit comparé à $\log N$ et $x_0 \in [0, 2^r)$ avec $2^r \ll N$ la graine secrète, il peut être défini de la façon suivante :

$$v_i = x_{i-1}^e \mod N \quad \text{et} \quad v_i = 2^k x_i + w_i \quad \text{pour} \quad i \geq 1,$$

où w_i représente la sortie de k bits du générateur à la i-ième itération.

Cette différence, bien qu'elle semble mineure, a un fort impact en terme de cryptanalyse. En effet, si la taille de la graine secrète est trop petite, la mise à la puissance e peut ne pas faire intervenir le module RSA. Dans ce cas, les équations engendrées par la relation de récurrence sont sur les entiers et retrouver la graine devient facile grâce à l'utilisation du lemme de Hensel. D'un autre côté, quand $r \geq \log N/e$, le problème devient difficile car même si la partie qui reste cachée x_i à chaque itération est relativement *petite*, les techniques de Coppersmith ne peuvent pas nous aider, la borne asymptotique étant justement la borne N/e.

Nous avons donc proposé dans un premier temps divers algorithmes de compromis temps-mémoire-données basés sur les tables de Hellman afin de récupérer une valeur d'état interne x_i. Ces algorithmes sont similaires à ceux utilisés dans les attaques contre les chiffrements par flot, et demandent donc de spécifier une fonction f. Pour les chiffrements pas flot, l'idée consiste à générer au moins $\log S$ bits pour un état interne de S bits. Dans notre cas, nous avons choisi de tronquer la sortie du générateur, ce qui peut sembler étonnant étant donné que l'itération de cette fonction n'a plus de lien avec la relation de récurrence, mais cela fonctionne car notre vrai besoin était de couvrir l'espace des valeurs possibles des états internes x_i. Il est important de préciser que nos algorithmes ne fonctionnent que pour le générateur Micali-Schnorr, grâce à son itération particulière, et ne peut s'appliquer aux générateurs que nous avons étudiés auparavant. Enfin, nous avons proposé différentes manières, après avoir retrouvé une valeur x_i, pour remonter jusqu'à la graine secrète. Tous ces algorithmes sont en complexité exponentielle ce qui limite bien évidemment leur utilisation mais permet de démontrer qu'une certaine marge de sécurité est à prendre par rapport à la borne d'application du lemme de Hensel.

Dans un second temps, nous avons réfléchi à une preuve de sécurité pour ce générateur. Celui-ci est prouvé sûr selon l'hypothèse relativement forte que la distribution de $x^e \mod N$, pour des entiers x de r bits, est indistinguable de la distribution uniforme des éléments de $(\mathbb{Z}/N\mathbb{Z})^*$ par tout test statistique en temps polynomial. Clairement cette hypothèse n'est pas vraie si l'on ne se retreint pas les tests à des temps polynomiaux étant donné le manque d'entropie apporté par l'entrée x. Micali et Schnorr ont proposé les paramètres $r = 2\log N/e$, ce qui permet d'éviter l'attaque trivial mais reste très agressif afin d'avoir une bonne efficacité.

Nous avons étudié les propriétés statistiques des k bits de poids faible de $x^e \mod N$ pour x étant choisi dans un petit intervalle $[0, M)$ avec $M < N$. Plus précisément, nous avons

prouvé deux bornes sur la distance statistique de ces k bits par rapport à la distribution uniforme en fonction de N, e, k, la première dans le cas où $M \gg \sqrt{N}$ et la seconde dans le cas opposé. Notons que nous avons considéré un cas général dans le sens où $\log M + k$ peut être strictement inférieur à la taille de l'état interne. Nous avons appliqué nos résultats sur une itération du générateur Micali-Schnorr (avec donc $\log M + k = \log N$) et avons trouvé une borne, dépendante de e et N, telle que la sortie de cette itération soit indistinguable de la distribution uniforme. Cette borne, asymptotique en N, implique qu'un résultat d'indistinguabilité est possible quand on sort moins de $1/3$ des bits de poids faible de l'état interne.

2.4 Perspectives et problèmes ouverts

Un certain nombre de ces travaux se sont conclus par des perspectives et/ou des problèmes ouverts qui sont détaillés ci-dessous.

En premier lieu, notre outil de recherche automatique d'attaques par faute est très récent et pourrait être approfondi, perfectionné et généralisé. Pour le moment, nous nous sommes concentrés sur un nombre restreint d'implémentations d'un nombre restreint de schémas tout en considérant peu de modèles de faute par rapport à tous ceux que l'on peut trouver dans la littérature. Un premier gros travail serait à effectuer pour obtenir une liste de toutes les conditions de fautes ayant mené à des attaques, et cela pour un maximum de schémas à clé publique, ce travail paraissant difficilement automatisable. De même, ajouter des modèles de faute demanderait un travail non négligeable surtout pour des modèles plus complexes. Enfin, il pourrait être intéressant de généraliser l'outil pour des implémentations de schémas à clé secrète telle que l'AES, ce qui demanderait probablement des conditions de faute très éloignées du format actuel.

Pour ce qui est de notre preuve de sécurité de notre schéma modifié RSA–PSS, il est clair que la suite consisterait à prendre en compte plus de types de faute mais aussi à donner à l'attaquant plus de latitude sur les étapes qu'il peut fauter puisque seul l'exponentiation modulaire peut être ciblée dans notre scénario actuel. Un autre problème à résoudre serait le suivant : actuellement le résultat de l'exponentiation modulo q fautée est une constante choisie par l'attaquant ; comment adapter la preuve dans le cas où ce résultat n'est plus forcément connu de l'attaquant mais s'il dépend de valeurs secrètes telles que la clé secrète ou le message encodé voir le résultat correct modulo q ?

Enfin, concernant le générateur Micali-Schnorr et notre étude de ses propriétés statistiques, nous insistons sur le fait que nous ne regardons qu'une seule itération comme cela a été fait dans d'autres travaux. Itérer sur plusieurs itérations consécutives notre borne sur la distance statistique semble aujourd'hui être un problème difficile.

2.5 Liste des publications

2.5.1 Articles de conférences

[BVZ12] Inferring Sequences Produced by Nonlinear Pseudorandom Number Generators Using Coppersmith's Methods
A. Bauer, D. Vergnaud and J.-C. Zapalowicz
PKC 2012

[FGL+12] Attacking RSA-CRT signatures with faults on Montgomery multiplication
P.-A. Fouque, N. Guillermin, D. Leresteux, M. Tibouchi and J.-C. Zapalowicz
CHES 2012

[FVZ13] Time/Memory/Data Tradeoffs for Variants of the RSA Problem
P.-A. Fouque, D. Vergnaud and J.-C. Zapalowicz
COCOON 2013

[FTZ13] Recovering Private Keys Generated With Weak PRNGs
P.-A. Fouque, M. Tibouchi and J.-C. Zapalowicz
IMACC 2013

[FZ14] Statistical properties of short RSA distribution and their cryptographic applications
P.-A. Fouque and J.-C. Zapalowicz
COCOON 2014

[BDF+14a] Making RSA-PSS Provably Secure Against Non-Random Faults
G. Barthe, F. Dupressoir, P-A. Fouque, B. Grégoire, M. Tibouchi and J-C. Zapalowicz
CHES 2014

[BDF+14b] Synthesis of Fault Attacks on Cryptographic Implementations
G. Barthe, F. Dupressoir, P.-A. Fouque, B. Grégoire and J.-C. Zapalowicz
ACM CCS 2014

[AFQ+14] Binary Elligator Squared
D. F. Aranha, P.-A. Fouque, C. Qian, M. Tibouchi and J.-C. Zapalowicz
SAC 2014

[AFG+14] GLV/GLS Decomposition, Power Analysis, and Attacks on ECDSA Signatures With Single-Bit Nonce Bias
D. F. Aranha, P.-A. Fouque, B. Gérard, J.-G. Kammerer, M. Tibouchi and J.-C. Zapalowicz
ASIACRYPT 2014

2.5.2 Articles de journal

[FGL+13] Attacking RSA-CRT signatures with faults on Montgomery multiplication
P.-A. Fouque, N. Guillermin, D. Leresteux, M. Tibouchi and J.-C. Zapalowicz
JCEN

PART I

FAULTS ON IMPLEMENTATIONS OF THE RSA SIGNATURE SCHEME

The RSA signature scheme is one of the most used schemes nowadays. An RSA signature is computed by applying some encoding function to the message, and raising the result to the d-th power modulo N, where d and N are the RSA private exponent and the RSA public modulus respectively. The modular exponentiation is the costliest part of the signature generation, so it is important to implement it efficiently. A very commonly used speed-up is RSA–CRT signature generation, where the exponentiation is carried out separately modulo the two factors of N, and the results are recombined using the Chinese Remainder Theorem (CRT). However, when unprotected, RSA–CRT signatures are vulnerable to the so-called Bellcore attack first introduced by Boneh, DeMillo and Lipton in [BDL97], and later refined in a number of subsequent publications [JLQ99, BDL01, YMH02, ABF+02]: an attacker who knows the padded message and is able to inject a fault in one of the two half-exponentiations can factor the public modulus using a faulty signature with a simple GCD computation.

Physical attacks. Embedded devices often play a central role in security architectures for large-scale software infrastructures. For instance, they are used pervasively for purposes such as authentication, identity management, and digital signatures. As a consequence, embedded devices are also a prime target for attackers. There are primarily two means to retrieve secret material from embedded devices. The first one is to carry non-invasive monitoring of the device and to obtain information from side-channels, such as timing and power consumption, electromagnetic radiations, or even noise. The second one is to perform active attacks, injecting faults that interfere with the normal execution of the devices, and to recover the secret information through the devices normal interface, or through side-channels. The effects of these faults vary: they may modify the control flow of the program by skipping a conditional test [AK97] or induce behaviours similar to buffer overflows [FLV12]. In the context of cryptographic attacks, they often allow the adversary to directly recover secret keys. There are multiple ways to inject faults in devices; examples include power spikes, glitches on the clock signal, temperature variations, and electromagnetic radiations [AK97, JT12, BBBP13].

The existence of efficient fault attacks against cryptographic schemes was first demonstrated in [BDL97], focusing the RSA–CRT implementation.

This first part is dedicated to some contributions in this prolific research area that is the impact of fault injection on implementations of the RSA signature scheme. Chapter 3 describe some new fault attacks against an implementation using the Chinese Remainder Theorem coupled with Montgomery multiplication. These attacks succeed whatever is the encoding function, using a single or a few faulted signatures depending on the scenario.

Chapter 4 is devoted to a new tool which automatically finds fault attacks given an implementation and the type of faults we authorize, which we call *fault conditions*. Finally, we consider in Chapter 5 RSA–PSS [BR98, BR01, Kal03], which is the RSA signature scheme with a specific probabilistic encoding function, and we propose a formally verified proof of the security of an infective countermeasure against a large class of non-random faults.

Contents

Chapter 3
Attacking RSA–CRT Signatures with Faults on Montgomery Multiplication

3.1 Introduction

In this chapter, we present several efficient fault attacks against implementations of RSA–CRT signatures that use modular exponentiation algorithms based on Montgomery multiplication. They apply to any padding function, including randomized paddings. The new attacks work provided that a small register can be forced to either zero, or a constant value, or a value with zero high-order bits. We show that these models are quite realistic, as such faults can be achieved against many proposed hardware designs for RSA signatures.

This work was presented at CHES 2012 [FGL+12] and appears in the *Journal of Cryptographic Engineering* [FGL+13].

3.1.1 Background

Many workarounds have been proposed to patch the vulnerability found by Boneh, DeMillo and Lipton in [BDL97], including extra computations and sanity checks of intermediate and final results. A recent taxonomy of these countermeasures is given in [Riv09]. The simplest countermeasure may be to verify the signature before releasing it. This is reasonably cheap if the public exponent e is small and available in the signing device. In some cases, however, e is not small, or even not given—e.g. the JavaCard API does not provide it [Ora]. Another approach is to use an extended modulus. Shamir's trick [Sha99] was the first such technique to be proposed; later refinements were suggested that also protect CRT recombination when it is computed using Garner's formula [BOS03, CJ05b, CGM+10, Vig08]. Finally, yet another way to protect RSA–CRT signatures against faults is to use redundant exponentiation algorithms, such as the Montgomery Ladder. Papers including [Gir06, Riv09] propose such countermeasures. Regardless of the approach, RSA–CRT fault countermeasures tend to be rather costly: for example, Rivain's countermeasure [Riv09] has a stated overhead of 10% compared to an unprotected implementation, and is purportedly more efficient than previous works including [Gir06, Vig08].

Relatedly, while Boneh et al.'s original fault attack does not apply to RSA signatures with probabilistic encoding functions, some extensions of it were proposed to attack randomized ad hoc padding schemes such as ISO 9796-2 and EMV [CJK+09, CNT10]. However, Coron and Mandal [CM09] were able to prove that Bellare and Rogaway's padding scheme RSA–PSS [BR98, BR01, Kal03] is secure against *random* faults in the random oracle model. In other words, if injecting a fault on the half-exponentiation modulo the second factor q of N produces a result that can be modeled as uniformly distributed modulo q,

then the result of such a fault cannot be used to break RSA–PSS signatures. It is tempting to conclude that using RSA–PSS should enable signers to dispense with costly RSA–CRT countermeasures.

3.1.2 Our contributions

The RSA–CRT implementations targeted in this chapter use the state-of-the-art modular multiplication algorithm due to Montgomery [Mon85], which avoids the need to compute actual divisions on large integers, replacing them with only multiplications and bit shifts. A typical implementation of the Montgomery multiplication algorithm will use small registers to store precomputed values or short integer variables throughout the computation. The size of these registers varies with the architecture, from a single bit in certain hardware implementations to 16 bits, 32 bits or more in software. This chapter presents several fault attacks on these small registers during Montgomery multiplication, that cause the result of one of the half-exponentiations to be unusually small. The factorization of N can then be recovered using a GCD, or an approximate common divisor algorithm such as [HG01, CN12, CH11].

We consider three models of faults on the small registers. In the first model, one register can be forced to zero. In that case, we show that causing such a fault in the inverse Montgomery transformation of the result of a half-exponentiation, or a few earlier consecutive Montgomery multiplications, yields a faulty signature which is a multiple of the corresponding factor q of N. Hence, we can factor N by taking a simple GCD. In the second model, another register can be forced to some (possibly unknown) constant value throughout the inverse Montgomery transformation of the result of a half-exponentiation, or a few earlier consecutive Montgomery multiplications. A faulty signature in this model is a close multiple of the corresponding factor q of N, and we can thus factor N using an approximate common divisor algorithm. Finally, the third model makes it possible to force some of the higher-order bits of one register to zero. We show that, while injecting one such fault at the end of the inverse Montgomery transformation results in a faulty signature that isn't usually close enough to a multiple of q to reveal the factorization of N on its own, a moderate number of faulty signatures (a dozen or so) obtained using that process are enough to factor N.

The RSA padding scheme used for signing, whether deterministic or probabilistic, is irrelevant in our attacks. In particular, RSA–PSS implementations are also vulnerable. Of course, this does not contradict the security result due to Coron and Mandal [CM09], as the faults we consider are strongly non-random. Our results do suggest, however, that exponentiation algorithms based on Montgomery multiplication are quite sensitive to a very realistic type of fault attacks and that using RSA–CRT countermeasures is advisable even for RSA–PSS.

3.1.3 Outline

In §3.2, we recall some background material on the Montgomery multiplication algorithm, on modular exponentiation techniques, and on RSA–CRT signatures. Our attacks are then described in §§3.3–3.5, corresponding to three different fault models: null faults, constant faults, and zero high-order bits faults. Finally, in §3.6, we discuss the applicability of our fault models to concrete hardware implementations of RSA–CRT signatures.

3.2 Preliminaries

3.2.1 Montgomery multiplication

First proposed by Montgomery in [Mon85], the Montgomery multiplication algorithm provides a fast method for computing modular multiplications and squarings. Indeed, the Montgomery multiplication algorithm only uses multiplications, additions and shifts, but no explicit division or modular reduction of big integers. Its cost is about twice that of a (non-modular) multiplication (compared to 2.5 times for a multiplication and a Barrett reduction), without any constraint on the modulus.

Usually, one of two different techniques is used to compute Montgomery multiplication: either Separate Operand Scanning (SOS), or Coarsely Integrated Operand Scanning (CIOS). Consider a device whose processor or coprocessor architecture has r-bit registers (typically $r = 1, 8, 16$, 32 or 64 bits). Let $b = 2^r$, q be the (odd) modulus with respect to which multiplications are carried out, k the number of r-bit registers used to store q, and $R = b^k$, so that $q < R$ and $\gcd(q, R) = 1$. The SOS variant consists in using the Montgomery reduction after the multiplication: for an input A such that $A < Rq$, it computes $\mathsf{Mgt}(A) \equiv AR^{-1} \pmod{q}$, with $0 \leq \mathsf{Mgt}(A) < q$. The CIOS mixes the reduction algorithm with the previous multiplication step: considering x and y with $xy < Rq$, it computes $\mathsf{CIOS}(x, y) = xyR^{-1} \bmod q$ with $\mathsf{CIOS}(x, y) < q$.

Algorithm 3.1 presents the main steps of the CIOS variant, which will be used thereafter. However, replacing the CIOS by the SOS or any other variant proposed in [KA96] does not protect against any of our attacks.

Algorithm 3.1 The Montgomery multiplication algorithm. The x_i's and y_i's are the digits of x and y in base b; $q' = -q^{-1} \bmod b$ is precomputed. The returned value is $(xy \cdot b^{-k} \bmod q)$. Since $b = 2^r$, the division is a bit shift.

1: **function** CIOS(x, y)
2: $a \leftarrow 0$
3: $y_0 \leftarrow y \bmod b$
4: **for** $j = 0$ to $k - 1$ **do**
5: $a_0 \leftarrow a \bmod b$
6: $u_j \leftarrow (a_0 + x_j \cdot y_0) \cdot q' \bmod b$
7: $a \leftarrow \left\lfloor \dfrac{a + x_j \cdot y + u_j \cdot q}{b} \right\rfloor$
8: **end for**
9: **if** $a \geq q$ **then** $a \leftarrow a - q$
10: **end if**
11: **return** a
12: **end function**

Among all the variants proposed for this algorithm, the optimization of [Wal99] is well-known: if $Rq > xy$, then the result of algorithm 3.1 without the final reduction (Step 9) is between 0 and $2q$. Therefore for an exponentiation algorithm, there is no need to carry out this final reduction if $R > 4q$. Besides its efficiency, this variant has the advantage of thwarting timing attacks [Sch00a, BB05, AScKK05], which essentially rely on detecting whether the reduction is carried out or not. Nevertheless, these attacks do not easily work with randomized paddings, since the attacker needs to carefully choose the message. In contrast, our attacks work on any padding, with or without this reduction step.

3.2.2 Exponentiation algorithms using Montgomery multiplication

Montgomery reduction is especially interesting when used as part of a modular exponentiation algorithm. Many such exponentiation algorithms can be found in the literature, including the Square-and-Multiply algorithm from either the least or the most significant bit of the exponent, the Montgomery Ladder (which is used as a side-channel countermeasure against cache analysis, branch analysis, timing analysis and power analysis), the Square-and-Multiply k-ary algorithm (which boasts greater efficiency thanks to fewer multiplications) and the Sliding Window algorithm. The previous five exponentiation algorithms will be considered in this chapter, and each of them (except the Square-and-Multiply MSB) is detailed in Figure 3.1.

Note that using the Montgomery multiplications inside any exponentiation algorithm requires all variables to be in Montgomery representation ($\bar{x} = xR \bmod q$ is the Montgomery representation of x) before applying the exponentiation process. In Step 2 of each algorithm from Figure 3.1, the message is transformed into Montgomery representation by computing $\mathsf{CIOS}(x, R^2) = xR^2R^{-1} \bmod q = \bar{x}$. At the end, the very last CIOS call allows to revert to the classical representation by performing a Montgomery reduction, i.e. $\mathsf{CIOS}(\bar{A}, 1) = (\bar{A} \cdot 1)R^{-1} \bmod q = ARR^{-1} \bmod q = A$. Finally the other CIOS steps compute the product in Montgomery representation: $\mathsf{CIOS}(\bar{A}, \bar{B}) = (AR)(BR)R^{-1} \bmod q = \overline{AB}$.

3.2.3 RSA–CRT signature generation

Let $N = pq$ be a n-bit RSA modulus. The public key is denoted by (N, e) and the associated private key by (p, q, d). For a message M to be signed, we note $S = m^d \bmod N$ the corresponding signature, where m is deduced from M by an encoding function, possibly randomized. A well-known optimization of this operation is RSA–CRT, which takes advantage of the decomposition of N into prime factors. By replacing a full exponentiation of size n by two $n/2$, it divides the computational cost by a factor of about 4. As a result, almost all implementations of RSA signatures use RSA–CRT, including OpenSSL [The] and the JavaCard API [Ora].

Recovering S from its reductions S_p and S_q modulo p and q can be done either by the usual CRT reconstruction formula (3.1) below, or using the recombination technique (3.2) due to Garner [Gar59]:

$$S = (S_q \cdot p^{-1} \bmod q) \cdot p + (S_p \cdot q^{-1} \bmod p) \cdot q \bmod N. \tag{3.1}$$

$$S = S_q + q \cdot (q^{-1} \cdot (S_p - S_q) \bmod p). \tag{3.2}$$

Garner's formula (3.2) does not require a reduction modulo N, which is interesting for efficiency reasons and also because it prevents certain fault attacks [BNNT11a]. On the other hand, it does require an inverse Montgomery transformation $S_q = \mathsf{CIOS}(\bar{S}_q, 1)$, whereas that step is not necessary for formula (3.1), as it can be mixed with the multiplication with $p^{-1} \bmod q$. This is an important point, as some of our attacks specifically target the inverse Montgomery transformation. The main steps of the RSA–CRT signature generation with Garner's recombination are recalled in Figure 3.2.

3.3 Null Faults

We first consider a fault model in which the attacker can force the register containing the precomputed value $q' = (-q^{-1} \bmod b)$ to zero in certain calls to the CIOS algorithm during the computation of S_q.

(a) Square-and-Multiply LSB

```
1: function Exp_LSB(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     A ← R mod q
4:     for i = 0 to t do
5:         if e_i = 1 then
6:             A ← CIOS(A, x̄)
7:         end if
8:         x̄ ← CIOS(x̄, x̄)
9:     end for
10:    A ← CIOS(A, 1)
11:    return A
12: end function
```

(b) Montgomery Ladder

```
1: function Exp_Ladder(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     A ← R mod q
4:     for i = t down to 0 do
5:         if e_i = 0 then
6:             x̄ ← CIOS(A, x̄)
7:             A ← CIOS(A, A)
8:         else if e_i = 1 then
9:             A ← CIOS(A, x̄)
10:            x̄ ← CIOS(x̄, x̄)
11:        end if
12:    end for
13:    A ← CIOS(A, 1)
14:    return A
15: end function
```

(c) Square-and-Multiply k-ary

```
1: function Exp_k-ary(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     x̄_0 ← R mod q
4:     for i = 1 to (2^k − 1) do
5:         x̄_i ← CIOS(x̄_{i−1}, x̄)
6:     end for
7:     A ← R mod q
8:     for i = t down to 0 do
9:         repeat k times
10:            A ← CIOS(A, A)
11:        A ← CIOS(A, x̄_{e_i})        ▷ (e in base 2^k)
12:    end for
13:    A ← CIOS(A, 1)
14:    return A
15: end function
```

(d) Sliding Window

```
1: function Exp_Sliding-Window(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     x̄_1 ← x̄;   x̄_2 ← CIOS(x̄, x̄)
4:     for i = 1 to (2^{k−1} − 1) do
5:         x̄_{2i+1} ← CIOS(x̄_{2i−1}, x̄_2)
6:     end for
7:     A ← R mod q;   i ← t
8:     while i ≥ 0 do
9:         if e_i = 0 then
10:            A ← CIOS(A, A);   i ← i − 1
11:        else if e_i = 1 then
12:            l ← min{j | j ≥ 0, i − j + 1 ≤ k and e_j = 1}
13:            repeat i − l + 1 times
14:                A ← CIOS(A, A)
15:            A ← CIOS(A, x̄_{e_i e_{i−1} ··· e_l})
16:            i ← l − 1
17:        end if
18:    end while
19:    A ← CIOS(A, 1)
20:    return A
21: end function
```

Figure 3.1: Four of the exponentiation algorithms considered in this chapter. In each case, $e_0, \ldots, e_t$ are the bits (or in the k-ary case, the digits in base 2^k) of the exponent e from the least to the most significant, and R is the Montgomery coefficient.

Algorithm 3.2 RSA–CRT signature generation with Garner's recombination. The reductions d_p, d_q modulo $p-1, q-1$ of the private exponent are precomputed, as is $\pi = q^{-1} \bmod p$.

function $\text{SIGN}_{\text{RSA–CRT}}(m)$
  $M \leftarrow \mu(m) \in \mathbb{Z}_N$ ▷ message encoding
  $M_p \leftarrow M \bmod p$
  $M_q \leftarrow M \bmod q$
  $S_p \leftarrow M_p^{d_p} \bmod p$
  $S_q \leftarrow M_q^{d_q} \bmod q$
  $t \leftarrow S_p - S_q$
  if $t < 0$ **then** $t \leftarrow t + p$
  end if
  $S \leftarrow S_q + \big((t \cdot \pi) \bmod p\big) \cdot q$
  return S
end function

Under suitable conditions, we will see that such faults can cause the q-part of the signature to be erroneously evaluated as $\widetilde{S}_q = 0$, which makes it possible to retrieve the factor q of N from one such faulty signature $\widetilde{S}$, as $q = \gcd(\widetilde{S}, N)$.

3.3.1 Attacking CIOS$(A, 1)$

Suppose first that the fault attacker can force q' to zero in the very last CIOS computation during the evaluation of S_q, namely the computation of CIOS$(A, 1)$. In that case, the situation is quite simple.

Theorem 3.1. *A faulty signature $\widetilde{S}$ generated in this fault model is a multiple of q (for any of the exponentiation algorithms considered herein and regardless of the encoding function involved, probabilistic or not).*

Proof. The faulty value $\widetilde{q'} = 0$ causes all of the variables u in the CIOS loop to vanish; indeed, for $j = 0, \ldots, k-1$, they evaluate to:

$$\widetilde{u}_j = (a_0 + A_j \cdot 1) \cdot \widetilde{q'} \bmod 2^r = 0.$$

As a result, the value $\widetilde{S}_q$ computed by this CIOS loop can be written as:

$$\widetilde{S}_q = \left\lfloor \left(\left\lfloor \cdots \left\lfloor \left(\lfloor A_0 \cdot 2^{-r} \rfloor + A_1 \right) \cdot 2^{-r} \right\rfloor + \cdots \right\rfloor + A_{k-1} \right) \cdot 2^{-r} \right\rfloor.$$

Now, the values A_j are r-words, i.e. $0 \leq A_j \leq 2^r - 1$. It follows that each of the integer divisions by 2^r evaluate to zero, and hence $\widetilde{S}_q = 0$. As a result, the faulty signature $\widetilde{S}$ is a multiple of q as stated. ■

It is thus easy to factor N with a single faulty signature $\widetilde{S}$, by computing $\gcd(\widetilde{S}, N)$. Note also that if this last CIOS step is computed as CIOS$(1, A)$ instead of CIOS$(A, 1)$, the formulas are slightly different but the result still holds.

3.3.2 Attacking consecutive CIOS steps

If Garner recombination is not used or the computation of CIOS$(A, 1)$ is somehow protected against faults, a similar result can be achieved by forcing q' to zero in earlier calls to CIOS, provided that a certain number of successive CIOS executions are faulty.

We consider here the fault model where we force q' to zero on consecutive CIOS steps. We will examine how this plays out in each on the five exponentiation algorithms in turn. For now, we let $\ell = \lceil \log_2 \lceil \log_2 q \rceil \rceil$.

In all cases, we also assume, heuristically, that the values $\bar{x}, A$ in Montgomery representation involved in our computation are uniformly distributed modulo q before the first fault is injected; this means, in particular, that they are smaller than $2^{\lceil \log_2 q \rceil}$ with probability at least $1/2$.

Square-and-Multiply LSB

We first consider a fault model in which the attacker can force the precomputed value q' to zero during ℓ consecutive calls of $\mathsf{CIOS}(\bar{x}, \bar{x})$ in the Square-and-Multiply LSB algorithm (Step 8 in Figure 3.1(a)), during the computation of S_q. Then, we claim that with probability at least $1/2$, a faulty signature $\widetilde{S}$ generated in this fault model will be a multiple of q, leading to the same key recovery as before.

Indeed, suppose faults are injected starting from iteration $i = \alpha$ in the loop of the exponentiation algorithm, and that before then, $|\bar{x}| \leq \lceil \log_2 q \rceil - 1$ where $|\bar{x}|$ represents the bit length: this happens with probability at least $1/2$. The fault $q' = 0$ has to occur in $\mathsf{CIOS}(\bar{x}, \bar{x})$ (the CIOS in Step 6 does not modify $\bar{x}$ and thus can be ignored in this case). Then, the output $\bar{x}$ of this faulty CIOS is, up to rounding errors:

$$\widetilde{x} \approx \left\lfloor \frac{\bar{x}_0 \bar{x}}{2^{rk}} \right\rfloor + \cdots + \left\lfloor \frac{\bar{x}_{k-1} \bar{x}}{2^r} \right\rfloor < \left\lfloor \frac{\bar{x}_{k-1} \bar{x}}{2^r} \right\rfloor + 2^{r(k-1)}.$$

With our assumption on the size of $\bar{x}$, we obtain $|\widetilde{x}| \leq \lceil \log_2 q \rceil - 2$. Therefore, for $i = \alpha + 1$ and with $q' = 0$, the size of the output of $\mathsf{CIOS}(\widetilde{x}, \widetilde{x})$ will be reduced to at most $\lceil \log_2 q \rceil - 4$, and so forth. By induction, keeping the fault $q' = 0$ up to iteration $i = \alpha + \ell - 1$, i.e. through ℓ executions of CIOS, brings the value $\bar{x}$ down to 0. Clearly, the faulty half-exponentiation thus outputs $\widetilde{S}_q = 0$, hence the stated result.

Square-and-Multiply MSB

The case of the Square-and-Multiply MSB exponentiation algorithm is similar. We claim that if we can force q' to zero during ℓ consecutive steps of the Square-and-Multiply MSB loop (hence up to 2ℓ calls to CIOS), then with probability at least $1/2$, a faulty signature $\widetilde{S}$ generated in this model will be a multiple of q.

The idea is again to target the Montgomery squaring step, which in the MSB case is $A \leftarrow \mathsf{CIOS}(A, A)$. The main difference with the LSB case is that the other CIOS also affects the same value A, and hence it is important that q' be kept to zero during this other call as well (typically, though, q' is simply kept to zero throughout sufficiently many loops, so this does not really matter in practice). Otherwise, the analysis is the same as in the LSB case, and we find that ℓ consecutive faulty iterations are sufficient to have $\widetilde{S}_q = 0$ with probability at least $1/2$.

Remark. While a fault has to be injected during the execution of $A \leftarrow \mathsf{CIOS}(A, \bar{x})$ as well, this faulty execution has some probability of reducing the size of A too, if the size of $\bar{x}$ is less than $\lceil \log_2 q \rceil$. Hence fewer iterations will be required in practice to reach $A = 0$. Moreover, if faults are injected throughout ℓ loops of the Square-and-Multiply MSB algorithm *starting from the very beginning*, we should get $\widetilde{S}_q = 0$ with probability 1. Indeed, the initial value of A is equal to $R \bmod q$, and hence is less than $\lceil \log_2 q \rceil$ bits.

Montgomery Ladder

In the case of the Montgomery Ladder, we obtain a similar result when q' is forced to zero in $2\ell - 1$ suitable consecutive iterations: we then obtain, again, that $\widetilde{S}_q = 0$ with probability at least $1/2$.

Indeed, a sequence of $2\ell - 1$ iterations of the Montgomery Ladder contains either at least ℓ Montgomery squarings of $\bar{x}$ (in Step 10 of Figure 3.1(b)) or ℓ Montgomery squarings of A (in Step 7, ibid.), depending on whether the corresponding string of bits of e contains more ones or zeros. The same analysis as in the Square-and-Multiply case shows that keeping q' to zero throughout the iterations containing those ℓ Montgomery squarings will have a probability at least $1/2$ of canceling $\bar{x}$ or A respectively, and hence yield $\widetilde{S}_q = 0$.

Remark. The stated bound of $2\ell - 1$ iterations is quite pessimistic, and the attack performs significantly better in practice, as both $\bar{x}$ and A will tend to become shorter during faulty Montgomery squarings, and hence both contribute to bringing each other's size further down in the CIOS calls that are not Montgomery squarings (Steps 6 and 9). Additionally, if faults are injected starting from the beginning of the loop, then, like in the Square-and-Multiply MSB setting, the attack succeeds with probability 1 in view of the initial value of A.

Square-and-Multiply k-ary

For the Square-and-Multiply k-ary exponentiation, forcing q' to zero during $\lceil \ell/k \rceil$ consecutive iterations of the loop is enough to achieve a probability of $1/2$ of getting $\widetilde{S}_q = 0$.

Indeed, the analysis is identical to the Square-and-Multiply MSB case of §3.3.2, except that each loop contains k Montgomery squarings instead of only one.

Sliding Window

Finally, we consider the case where q' can be forced to zero in consecutive iterations of the main loop (Steps 8–18 in Figure 3.1(d)) of the Sliding Window exponentiation algorithm, and claim that ℓ consecutive faulty iterations are enough to obtain $\widetilde{S}_q = 0$ with probability $1/2$.

Again, the situation is analogous to the Square-and-Multiply MSB setting of §3.3.2, in the sense that each iteration of the loop contains at least one Montgomery squaring of A, hence ℓ iterations are enough to reach $A = 0$.

Simulation results

We have carried out a simulation of null faults on consecutive CIOS steps for the three first exponentiation process algorithms, with varying numbers of faulty iterations; for the Square-and-Multiply MSB and the Montgomery Ladder algorithms, two sets of experiments have been conducted for each parameter set: one with faults starting from the first iteration, and another one with faults starting from a random iteration somewhere in the exponentiation loop. Results are collected in Table 3.1. As we can see, success rates are noticeably higher than those claimed above. This is due to the fact that, with probability $1/2$ at each faulted step of the exponentiation, the size of the focused value can decrease more than the theoretical minimum.

Table 3.1: Success rate of the null fault attack on consecutive CIOS steps, for a 512-bit prime q and $r = 16$. 100 faulty signatures were computed for each parameter set. For the Square-and-Multiply MSB and Montgomery Ladder algorithms, we compare success rates when faults start at the beginning of the loop vs. at a random iteration.

	S&M LSB	S&M MSB		Montgomery Ladder	
Faulty iterations	(%)	Start (%)	Anywhere (%)	Start (%)	Anywhere (%)
8	31	93	62	45	30
9	65	100	93	87	76
10	89	100	100	99	93

3.4 Constant Faults

In this section, we consider a different fault model, in which the fault attacker can force the variables u_j in the CIOS algorithm to some (possibly unknown) constant value $\widetilde{u}$.

Just as with null faults, we consider two scenarios: one in which the last CIOS computation is attacked, and another in which several inner consecutive CIOS computations in the exponentiation algorithm are targeted.

3.4.1 Attacking $\mathsf{CIOS}(A, 1)$

Faults on all iterations. Consider first the case when faults are injected in all iterations of the very last CIOS computation. In other words, the device will compute $\mathsf{CIOS}(A, 1)$, except that the variables u_j, $j = 0, \ldots, k-1$, are replaced by a fixed, possibly unknown value $\widetilde{u}$. In that case, we show that a single faulty signature is enough to factor N and recover the secret key. The key result is as follows:

Theorem 3.2. *Let $\widetilde{S}$ be a faulty signature obtained in the fault model described above. Then, $(2^r - 1) \cdot \widetilde{S}$ is a close multiple of q with error size at most 2^{r+1}, i.e. there exists an integer T such that:*

$$\left|(2^r - 1) \cdot (\widetilde{S} + 1) - qT\right| \leq 2^{r+1}.$$

Proof. Up to the possible subtraction of q, which clearly doesn't affect our result, the value $\widetilde{S}_q$ computed in the faulty execution of $\mathsf{CIOS}(A, 1)$ can be written as:

$$\widetilde{S}_q = \left\lfloor \left(\left\lfloor \cdots \left\lfloor (A_0 + \widetilde{u} \cdot q) \cdot 2^{-r} \right\rfloor + \cdots \right\rfloor + A_{k-1} + \widetilde{u} \cdot q \right) \cdot 2^{-r} \right\rfloor.$$

We claim that this value $\widetilde{S}_q$ is close to the real number $\widetilde{u} \cdot q/(2^r - 1)$. Indeed, on the one hand, by using the fact that $\lfloor x \rfloor \leq x$ for all x and $A_j \leq 2^r - 1$ for $j = 0, \ldots, k-1$, we obtain:

$$\begin{aligned} \widetilde{S}_q &\leq \frac{A_0 + \widetilde{u} \cdot q}{2^{rk}} + \cdots + \frac{A_{k-1} + \widetilde{u} \cdot q}{2^r} \\ &\leq \frac{1}{2^r - 1}(2^r - 1 + \widetilde{u} \cdot q) \leq \frac{\widetilde{u} \cdot q}{2^r - 1} + 1. \end{aligned}$$

On the other hand, since $\lfloor x \rfloor > x - 1$ and $A_j \geq 0$, we get:

$$\begin{aligned}\widetilde{S}_q &> \frac{\widetilde{u} \cdot q}{2^{rk}} - \frac{1}{2^{r(k-1)}} + \cdots + \frac{\widetilde{u} \cdot q}{2^r} - 1 \\ &> \frac{1 - 2^{-rk}}{2^r - 1} \cdot (\widetilde{u} \cdot q - 2^r) \\ &> \frac{\widetilde{u} \cdot q}{2^r - 1} - \frac{\widetilde{u}}{2^r - 1} \cdot \frac{q}{2^{rk}} - \frac{2^r}{2^r - 1} > \frac{\widetilde{u} \cdot q}{2^r - 1} - 3\end{aligned}$$

as we have both $\widetilde{u} \leq 2^r - 1$ and $q < 2^{rk}$. As a result, we obtain:

$$\left| (\widetilde{S}_q + 1) - \frac{\widetilde{u} \cdot q}{2^r - 1} \right| \leq 2$$

and hence:

$$\left| (2^r - 1) \cdot (\widetilde{S}_q + 1) - \widetilde{u} \cdot q \right| \leq 2^{r+1}.$$

Since $\widetilde{S} = \widetilde{S}_q + ((\tilde{t} \cdot \pi) \bmod p) \cdot q$, the stated result follows, with $T = \widetilde{u} + (2^r - 1) \cdot ((\tilde{t} \cdot \pi) \bmod p)$. ■

Thus, a single faulty signature yields a value $V = (2^r - 1) \cdot (\widetilde{S} + 1) \bmod N$ which is very close to a multiple of q. It is easy to use this value to recover q itself. Several methods are available:

- If r is small (say 8 or 16), it may be easiest to just use exhaustive search: q is found among the values $\gcd(V + X, N)$ for $|X| \leq 2^{r+1}$, and hence can be retrieved using around 2^{r+2} GCD computations.
- A more sophisticated option, which may be interesting for $r = 32$, is the baby step, giant step-like algorithm by Chen and Nguyen [CN12], which runs in time $\widetilde{O}(2^{r/2})$.
- Alternatively, for any r up to half of the size of q, one can use Howgrave-Graham's algorithm [HG01] based on Coppersmith techniques. It is the fastest option unless r is very small (a simple implementation in Sage [S+14] runs in about 1.5 ms on our standard desktop PC with a 512-bit prime q for a any r up to ≈ 160 bits, whereas exhaustive search already takes over one second for $r = 16$).

Faults on most iterations. In fact, Howgrave-Graham's algorithm is especially relevant if the constant faults do not start at the very first iteration in the CIOS loop. More precisely, suppose that the fault attacker can force the variables u_j to a constant value $\widetilde{u}$ not for all j but for $j = j_0, j_0 + 1, \ldots, k - 1$ for some j_0.

Then, the same computation as in the proof of Theorem 3.2 yields the following bound on $\widetilde{S}_q$:

$$\frac{\widetilde{u} \cdot q}{2^r - 1} - 2^{rj_0} - 2 < \widetilde{S}_q \leq \frac{\widetilde{u} \cdot q}{2^r - 1} + 2^{rj_0} + 1.$$

It follows that $(2^r - 1) \cdot \widetilde{S}$ is a close multiple of q with error size $\lesssim 2^{r(j_0+1)}$.

Now note that Howgrave-Graham's algorithm [HG01] will recover q given N and a close multiple with error size at most $q^{1/2-\varepsilon}$. This means that one faulty signature $\widetilde{S}$ is enough to factor N as long as $j_0 + 1 < k/2$, i.e. the constant faults start in the first half of the CIOS loop.

Moreover, if the faults start even later, a single signature will no longer suffice, but q can be recovered if several faults are available, by using the generalization of Howgrave-Graham's algorithm due to Cohn and Heninger [CH11]. That algorithm is discussed in a different context in §3.5.

3.4.2 Attacking other CIOS steps

As in §3.3.2, if Garner recombination is not used or if $\mathsf{CIOS}(A, 1)$ is protected against faults, we can adapt the previous attack to target earlier calls to CIOS and still reveal the factorization of N. However, the attack requires two faulty signatures with the same constant fault $\widetilde{u}$.

For now, we focus on the Square-and-Multiply LSB algorithm and assume that constant faults are injected in the evaluations of $\mathsf{CIOS}(\bar{x}, \bar{x})$ and $\mathsf{CIOS}(A, \bar{x})$ during the exponentiation process computing S_q. More precisely, suppose that $u_i = \widetilde{u}$ $(i = 0, \ldots, k-1)$ for these particular CIOS, and write:

$$\ell = 2^{\lceil \log_2 q \rceil - 1} \sqrt{1 - \frac{\widetilde{u}}{(2^r - 1)2^{\lceil \log_2 q \rceil - 2}}}.$$

We claim that if the initial value of $\bar{x}$ is such that $\ell < \bar{x} < 2^{\lceil \log_2 q \rceil - 1}$, then the computed value A of the Square-and-Multiply LSB approaches ℓ.

Indeed, one can see that the output $\widetilde{x}$ of each faulty $\mathsf{CIOS}(\bar{x}, \bar{x})$ is roughly:

$$\widetilde{x} \approx \frac{\bar{x}^2}{2^{\lceil \log_2 q \rceil}} + \frac{\widetilde{u}q}{2^r - 1} - \varepsilon \cdot q \quad (\varepsilon \in \{0, 1\})$$

the variable ϵ being necessary since the result of algorithm 3.1 without the final reduction (Step 9) is between 0 and $2q$. Looking at the sequence $v_{n+1} = f(v_n) = av_n^2 + c$ with $a = \frac{1}{2^{\lceil \log_2 q \rceil}}$, $c = \frac{\widetilde{u}q}{2^r - 1}$ and v_0 represents the message in Montgomery representation. This sequence will tend to a limit ℓ if $\Delta = 1 - 4ac > 0$, i.e. $\frac{\widetilde{u}}{2^r - 1} < \frac{2^{\lceil \log_2 q \rceil - 2}}{q}$. Our assumption on the value of v_0 implies that $f(I) \in I$. Referring to the graph below, it appears then that the sequence will tend to $\ell = \min(\ell_1, \ell_2)$ where ℓ_1 and ℓ_2 denote the two roots of $f(\ell) = \ell$.

However, we want that this limit be reached before the end of the exponentiation process. Let us determine the convergence speed of this sequence:

$$\begin{aligned} |v_{n+1} - \ell| &= |f(v_n) - f(\ell)| \leq |f'(\ell)| \cdot |v_n - l| \\ &\leq |f'(\ell)|^{n+1} \cdot |v_0 - \ell| \end{aligned}$$

Since ℓ and v_0 are integer values, we look for the condition $|f'(\ell)|^{n+1} \cdot |v_0 - \ell| \leq 1$. Hence, the limit is reached for n such that:

$$n + 1 \geq -\frac{\log_2(|v_0 - \ell|)}{\log_2(|f'(\ell)|)}$$

For example, we search a condition in order to have $\mathsf{CIOS}(\bar{x}, \bar{x}) = \ell$ before the half $(|q|/2)$ of the exponentiation process. Since $\log_2(|v_0 - \ell|) \approx |q|$, this condition is $\frac{-1}{\log_2(|f'(\ell)|)} < 1/2$, i.e. $|f'(\ell)| < 1/4$. Moreover, $f'(\ell) = 2a\ell = 1 - \sqrt{\Delta} > 0$ leads to $9/16 < \Delta$ and then to $\frac{\widetilde{u}}{2^r - 1} < \frac{9}{16} \frac{2^{\lceil \log_2 q \rceil - 2}}{q}$. We see on this example that the success of the attack will depend on the ratio $q/2^{\lceil \log_2 q \rceil}$ and on the ratio $\widetilde{u}/(2^r - 1)$.

Looking at $\mathsf{CIOS}(A, \bar{x})$, the output $\widetilde{A}$ is roughly:

$$\widetilde{A} \approx \frac{A\bar{x}}{2^{\lceil \log_2 q \rceil}} + \frac{\widetilde{u}}{2^r - 1} - \varepsilon \cdot q \quad (\varepsilon \in \{0, 1\})$$

and the associated sequence is a little more complicated:

$$g(w_n) = w_{n+1} = \begin{cases} w_n & \text{if } e_{n+1} = 0 \\ aw_n v_n + c & \text{else} \end{cases}$$

Table 3.2: Success rate of the constant fault attack on successive CIOS steps, when using Square-and-Multiply LSB exponentiation with random 512-bit primes q and $r = 16$.

$q/2^{\lceil \log_2 q \rceil}$	0.666	0.696	0.846	0.957
Success rate (%)	36	34.4	26.7	20.4

It is clear that if $v_n = \ell$, w_n will tend to this limit too. We just have to verify that this sequence reaches ℓ before the end of the exponentiation process. In fact, both sequences are linked by the following relation:

$$\frac{w_{n+1} - v_{n+1}}{w_n - v_n} = \frac{v_n}{2^{\lceil \log_2 q \rceil}}$$

With our assumption, the sequence (v_n) is decreasing, and we have:

$$\frac{w_{\lceil \log_2 q \rceil} - v_{\lceil \log_2 q \rceil}}{w_0 - v_0} < \frac{1}{2^{\lceil \log_2 q \rceil}}$$

Hence the range between the two sequences constantly decreases during the exponentiation process and if (v_n) tends to ℓ before the end of the exponentiation process, then (w_n) will reach this value too.

The attack consists on computing two signatures $\tilde{S}, \tilde{S}'$ by faulting them with the same fault $\widetilde{u}$. In consequence, with a certain probability depending on the ratios $q/2^{\lceil \log_2 q \rceil}$ and $\widetilde{u}/(2^r - 1)$, these two signatures will be equal modulo q. Thus, we recover q as $\gcd(N, \tilde{S} - \tilde{S}')$. This attack works with the Square-and-Multiply LSB and Montgomery Ladder algorithms, but not with the three other exponentiations.

Remark. In Table 3.2, the success rates are even better than expected. In fact, if $\Delta < 0$, the value $\bar{x}$ can enter in a cycle of a few different values. As a consequence, with some probability, two messages can have the same value S_q. Graphically, that can be explained by the representation of the function $f \circ \cdots \circ f$ which is flatter and can intersect the line representing $g(x) = x$.

Simulation results are presented in Table 3.2. For various 512-bit primes q, the attack has been carried out for 1000 pairs of random messages, with a random constant fault $\widetilde{u}$ for each pair. It is successful if the two resulting faulty signatures $\widetilde{S}$, $\widetilde{S}'$ satisfy $\gcd(N, \widetilde{S} - \widetilde{S}') = q$.

3.5 Zero High-Order Bits Faults

In this section, we consider yet another fault model, in which the fault attacker targets the very last iteration in the evaluation of $\mathsf{CIOS}(A, 1)$ during the computation of S_q. We assume that the attacker is able to force a certain number h of the highest-order bits of u_{k-1} to zero, possibly but not necessarily all of them (i.e. $1 \leq h \leq r$ and $u_{k-1} < 2^{r-h}$). Then, while a single faulty signature is typically not sufficient to factor the modulus, multiple such signatures will be enough if h is not too small.

Theorem 3.3. *Let $\widetilde{S}$ be a faulty signature obtained in this fault model. Then, $\widetilde{S}$ is a close multiple of q with error size at most $2^{-h} \cdot q + 1$, i.e. there exists an integer T such that $|\widetilde{S} - qT| \leq 2^{-h} \cdot q + 1$.*

Proof. The iterations numbered $0, 1, \ldots, k-2$ in the evaluation of $\mathsf{CIOS}(A, 1)$ are all computed correctly. Let $a_1, a_2, \ldots, a_{k-1}$ be the values of the variable a at the end of these respective iterations. We have:

$$\begin{aligned} a_1 &= \left\lfloor \frac{0 + A_0 + u_0 \cdot q}{2^r} \right\rfloor \\ &\leq \frac{0 + (2^r - 1) + (2^r - 1) \cdot q}{2^r} \leq q + 1 \\ a_2 &= \left\lfloor \frac{a_1 + A_1 + u_1 \cdot q}{2^r} \right\rfloor \\ &\leq \frac{(q+1) + (2^r - 1) + (2^r - 1) \cdot q}{2^r} \leq q + 1 \end{aligned}$$

and it is then easy to see by induction that $0 \leq a_{k-1} \leq q + 1$. Then, the computation of the last iteration is attacked, with a value $\widetilde{u}_{k-1}$ of u satisfying $0 \leq \widetilde{u}_{k-1} \leq 2^{r-h} - 1$. Thus, the value of a after that iteration becomes:

$$\begin{aligned} a_k &= \left\lfloor \frac{a_{k-1} + A_{k-1} + \widetilde{u}_{k-1} \cdot q}{2^r} \right\rfloor \\ &\leq \frac{(q+1) + (2^r - 1) + (2^{r-h} - 1) \cdot q}{2^r} \leq \frac{q}{2^h} + 1. \end{aligned}$$

In particular, $a_k < q$, so that the q-part of the signature $\widetilde{S}_q$ is a_k, and hence $|\widetilde{S}_q| \leq 2^{-h} \cdot q + 1$. Since $\widetilde{S}_q = \widetilde{S} - qT$ for $T = (t \cdot p) \bmod p$, this concludes the proof. ■

Note that exactly the same result still holds if, in addition to u_{k-1}, previous values of u are attacked in the same fashion as well, so there is no need to synchronize the attack extremely precisely so as to target only the last iteration.

Now, recovering q from faulty signatures of the form $\widetilde{S}$ is a partial approximate common divisor (PACD) problem, as we know one exact multiple of q, namely N, and several close multiples, namely the faulty signatures. Since the error size $\approx q/2^h$ is rather large relative to q, the state-of-the-art algorithm to recover q in that case is the one proposed by Cohn and Heninger [CH11] using multivariate Coppersmith techniques.

The algorithm by Cohn and Heninger is likely to recover the common divisor $q \approx N^{1/2}$ given ℓ close multiples $\widetilde{S}^{(1)}, \ldots, \widetilde{S}^{(\ell)}$ provided that the error size is significantly less than $N^{(1/2)^{1+1/\ell}}$. Thus, we should have:

$$\log_2 q - h \lesssim \left(\frac{1}{2}\right)^{1+1/\ell} \log_2 N \approx 2^{1/\ell} \cdot \log_2 q.$$

Hence, if the faults cancel the top h bits of u_{k-1}, we need ℓ of them to factor the modulus, where:

$$\ell \gtrsim -\frac{1}{\log_2\left(1 - \frac{h}{\log_2 q}\right)}. \tag{3.3}$$

In practice, if a few more faults can be collected, it is probably preferable to simply use the linear case of the Cohn-Heninger attack (the case $t = k = 1$ in their paper [CH11]), since it is much easier to implement (as it requires only linear algebra rather than Gröbner bases) and involves lattice reduction in a lattice of small dimension that is straightforward

Table 3.3: Theoretical minimum number ℓ of zero higher-order h-bit faulty signatures required to factor a balanced 1024-bit RSA modulus N using the general Cohn-Heninger attack or the simplified linear one.

Number h of zero top bits	48	40	32	24	16
Minimum ℓ with the general attack	8	9	11	15	22
Minimum ℓ with the linear attack	11	13	16	22	32

to construct. More precisely, reducing the lattice L generated by the rows of the following matrix:

$$\begin{pmatrix} B & & & -\widetilde{S}^{(1)} \\ & \ddots & & \vdots \\ & & B & -\widetilde{S}^{(\ell)} \\ & & & N \end{pmatrix}$$

where $B = 2^{kr-h}$, gives a lattice basis consisting of affine forms the first ℓ of which vanish on the vector of "error values" $\left(\widetilde{S}_q^{(1)}/B, \ldots, \widetilde{S}_q^{(\ell)}/B\right)$ if ℓ is large enough. More precisely, they vanish on this vector modulo q, and also do over the integers provided that their coefficients are much smaller than q. If we assume that L behaves like a random lattice, the length of vectors in the reduced basis should be roughly $\det(L)^{1/\dim(L)} = (B^\ell \cdot N)^{1/(\ell+1)}$. This gives the condition:

$$B^\ell \cdot N \ll q^{\ell+1}$$
$$\ell(\log_2 q - h) + \log_2 N \lesssim (\ell+1)\log_2 q.$$

Hence, this method should recover the error vector and thus make it possible to factor N provided that:

$$\ell \gtrsim \frac{\log_2 q}{h} \tag{3.4}$$

which is always a worse bound than (3.3) but usually not by a very large margin. Table 3.3 gives the theoretical number of faulty signatures required to factor N for various values of h, both in the general attack by Cohn and Heninger and in the simplified linear case.

We carried out a simulation of the linear version of the attack on a 1024-bit modulus N with various values of h, and found that it works very well in practice with a number of faulty signatures consistent with the theoretical minimum. The results are collected in Table 3.4. The attack is also quite fast: a naïve implementation in Sage runs in a fraction of a second on a standard PC.

3.6 Fault Models

In this section we discuss how realistic the setup of the attacks described above can be. In principle, all the RSA–CRT implementations using Montgomery multiplication may be vulnerable, but we have to note that the fault setup (and how realistic it is) depends heavily on implementation choices, since many variations around the algorithm from Figure 3.1

Table 3.4: Experimental success rate of the simplified (linear) Cohn-Heninger attack with ℓ faulty signatures when N is a balanced 1024-bit RSA modulus. Timings are given for our Sage implementation on a single core of a Core 2 CPU at 3 GHz.

Number ℓ of faulty signatures	11	12	13	14	15	16	17	18
Success rate with $h = 48$ (%)	23	100	100	100	100	100	100	100
Success rate with $h = 40$ (%)	0	0	2	100	100	100	100	100
Success rate with $h = 32$ (%)	0	0	0	0	0	0	99	100
Average CPU time (ms)	33	35	38	41	45	49	54	59

have been proposed in recent literature. After a discussion of the characteristics of the tools needed to get the desired effects, we focus on several implementation proposals [TcKK99, MMM04, HGKEG08, NMSiK01, Suz07, MSPV07, CELL10], chosen for their relevance, and discuss whether our fault model is realistic in those settings.

3.6.1 Characteristics of the perturbation tool

First, all the perturbations needed to carry out our attacks need to be controlled and local to some gates of the chip. Therefore, the attacker needs to identify the localization of the vulnerable gates and registers. The null fault attacks described in §3.3 need either a q' value set to 0, or multiple consecutive faults in Step 6 of the main loop of $\mathsf{CIOS}(A, 1)$ or during multiple consecutive CIOS. The attacks described in §3.4 also need these multiple consecutive faults. Considering that state-of-art secure micro-controllers embed desynchronization countermeasures such as clock jitters and idle cycles, if the target of the perturbation is some shared logic with other treatments (like in the ALU of a CPU), the fault must be accurately space and time controlled, and the effects must be repeatable as well. Identification of the good cycles for injecting the perturbation may be a very difficult task, and our attacks seem to be irrelevant. The only exception may be the null fault of §3.3, if the fault is injected when the q' register is loaded.

Nevertheless, many secure microcontrollers embed an isolated modular arithmetic acceleration coprocessor. A large proportion of them specifically use the Montgomery multiplication CIOS algorithm (or one of its described variants [KA96]). Therefore, if the q' or the u_j value is isolated in a specific small size register, a unique long duration perturbation can be sufficient for our attack to succeed. The duration of the perturbation varies with the implementation choices and can vary from one cycle to $\log_2 q$, which does not exceed a hundred microseconds on actual chips. To get this kind of effect, laser diodes are the best-suited tool, since the duration of the spot is completely controlled by the attacker [Sko10].

3.6.2 Analysis of classical implementations of the Montgomery multiplication

The Montgomery coprocessors proposed in the literature can be divided in 3 different categories:

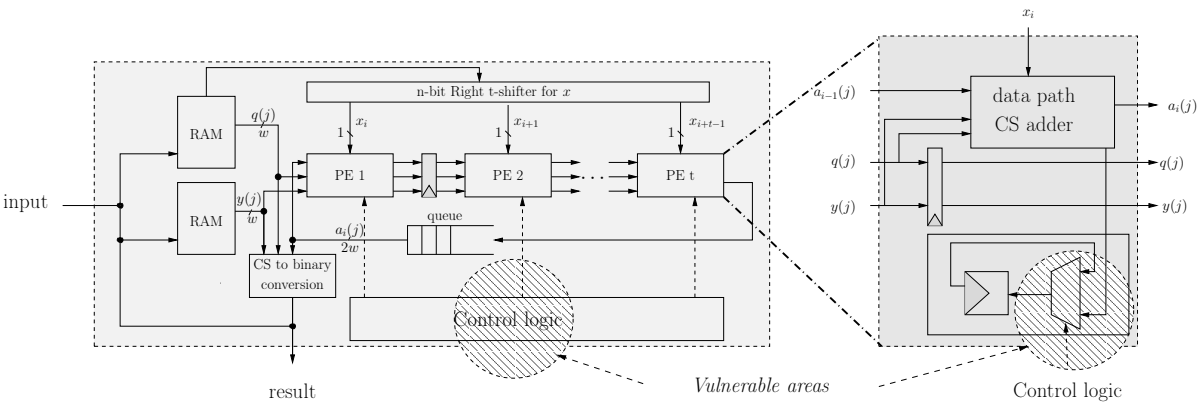

Figure 3.2: Systolic Montgomery Multiplier of [TcKK99] and potential fault target.

- the first category [TcKK99, MMM04, HGKEG08] contains variations on the Tenca and Koç *Multiple Word Radix-2 Montgomery Multiplication* algorithm (MWR2MM) [TcKK99], which can be seen as a CIOS algorithm with $r = 1$. The characteristic of these implementations is that they use no multiplier architecture, and are therefore really suitable for constrained ASIC implementations.
- the second category [Suz07, MSPV07] is an intermediate where r is a classical size for embedded architecture, such as 8, 16 or 32 bits, or even 36 bits if we consider FPGA architectures. These designs are better-suited for the FPGA setting, since these technologies embed very powerful built-in multiplier blocks. Nevertheless they can also be used in ASIC design for intermediate area/latency trade-offs.
- finally, the last category [NMSiK01, CELL10] proposes a version of CIOS/SOS with only one loop, implying that $r \geq \lceil \log_2 q \rceil$. The main difficulty of these implementation techniques is to deal with the very large multiplications they require (one $r \times r$ and two half $r \times r$ multiplications per CIOS). For that purpose they use interpolation techniques, like Karatsuba in [CELL10] or the *Residue Number System* (RNS) in [NMSiK01]. These implementations are designed to achieve the shortest latency, and are therefore area consuming.

Architectures based on MWR2MM: $r = 1$

In this kind of architecture, q' cannot be manipulated, since it is always equal to 1, so no wire or register carries its value. On the other hand, the value of u_j is computed at every loop of the CIOS, and since it is only one bit, a simple shot on the logic driving the register during the final multiplication CIOS$(A, 1)$ is sufficient to get an exploitable result ($u_j = 0$ corresponds to the null fault of §3.3, and $u_j = 1$ to the constant fault of §3.4).

The first proposal [TcKK99] is a fully systolic[1] array of processing elements (PE) executing consecutively line 6 of the CIOS algorithm in one cycle, and line 7 in k cycles from LSB to MSB. Figure 3.2 proposes an overview of the architecture. Each PE consists of a w-word carry save adder, able to compute a w word addition and to keep the carry for the next cycle. In the figure, $T(j)$ stands for the j-th least significant w word of T.

At each clock cycle, the PE presents the computed result $a_i(j)$ to the next one, and the value u_i is kept in the PE for the computation of the next word $a_i(j + 1)$. The

[1] Meaning that all the PEs are the same.

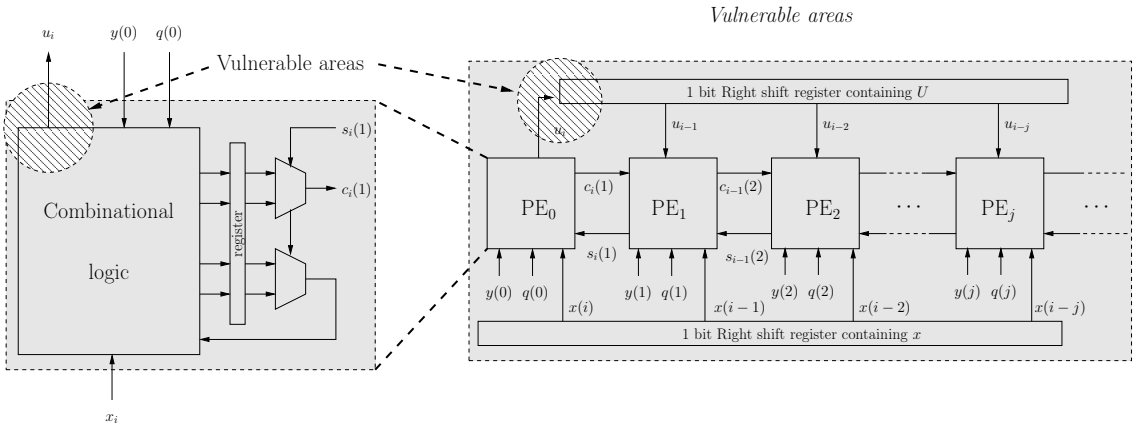

Figure 3.3: Overview of the architecture from [HGKEG08] and potential fault target.

value of u_i is computed before the word $a_i(0)$ is presented, and then is kept in each PE during the whole computation of a_i in a register. After a complete multiplication, the result a_n is transformed from a carry save representation to binary thanks to the CS to binary converter. This architecture has the great advantage of being completely scalable. Whatever the number of PEs and the size of M, this architecture can compute the expected result as long as the RAM are correctly dimensioned.

To achieve our attack, the register keeping u_i can be targeted, but every PE must be targeted simultaneously in order to get the correct result. Therefore it is more interesting to target the control logic responsible for the sequencing of the register loading, since all the PEs are connected.

In [MMM04], the authors manage to get rid of the CS to binary converter by redesigning the CS adder of every PE. The vulnerability to our attack is therefore the same, since the redesign does not affect the targeted area.

Huang et al. [HGKEG08] proposed a new version of the data dependency in the MWR2MM algorithm and rearranged the architecture of [TcKK99], in a semi systolic form. Figure 3.3 gives an overview of the architecture. In this architecture, the intermediate value a_i is manipulated in carry save format ($a_i = c_i + s_i$). A specific PE, PE$_0$ is specialized in generating the u_i values at each cycle, while the j-th PE is in charge of computing the sequence $a_i(j)$. The scalability is lost in exchange for a better time/area trade-off.

This architecture is very vulnerable to our attacks, since a simple n-cycle long shot on the right logic in PE$_0$ (see Figure 3.3) is sufficient to get the expected result.

According to the authors, the design works at 100 MHz on their target platform (a Xilinx Virtex II FPGA), therefore the duration of the perturbation is at least 10 μs for a 1024 bits multiplication (2048 bits RSA) if the Garner recombination is used (using the attack from §3.3.1 or §3.4.1). If classical CRT reconstruction is used, according to Table 3.1, 200 μs will be enough for a null fault.

As a conclusion we can see that this kind of implementation is very vulnerable, since the setup of the attack is quite simple.

High radix architecture: $1 < r < \lceil \log_2 q \rceil$

In this type of implementation choice the value $q' = -q^{-1} \bmod 2^r$ is computed in a r-bit register, unless the quotient pipelining approach [Oru95] is used. In all the implementations, the value q' is an r-bit register and can be the target of the attack.

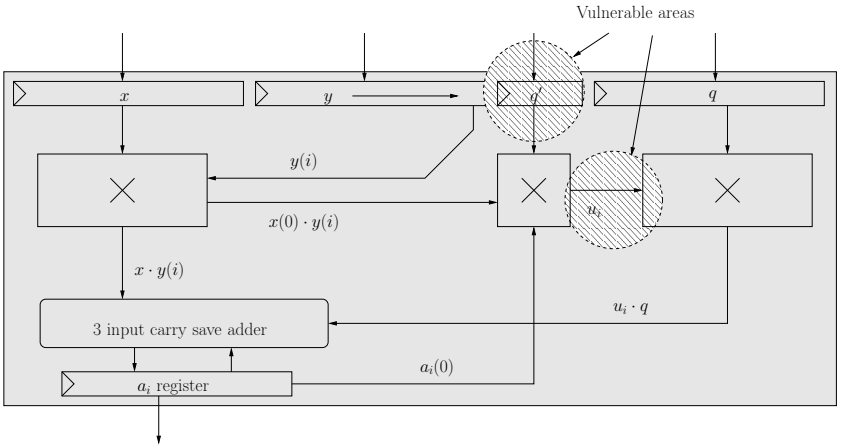

Figure 3.4: Overview of the architecture from [MSPV07] and potential fault target.

For example, the implementation of [MSPV07] is described in Figure 3.4. It relies on the coordinated usage of multiplier blocks of the Xilinx Virtex II together with specifically designed carry save adders. The CIOS algorithm from Figure 3.1 is completely respected in this implementation. The values u_j can be the target of any fault described in this chapter, but it may be easier to put once for all the q' register to 0, with a 100% success rate for the attack if properly carried out. Another implementation is mentioned in [MSPV07] with a four-deep pipeline, but it suffers from the same vulnerability.

On the other hand, the attack may be more difficult to achieve on the architecture described in [Suz07, Figure 4]. First, it uses quotient determination [Oru95], and therefore does not need to store q' anywhere. Second, the multiplier in charge of computing u_j is shared for all the Montgomery computation. In order to carry out the attack of §3.4 on this architecture, the attacker has to determine the specific cycles where u_j is computed to generate a perturbation. For that particular design, the attacks seem out of reach.

Full radix architecture: $r \geq \lceil \log_2 q \rceil$

In this kind of implementation, a single round is enough to compute the Montgomery algorithm. This implementation choice concentrates all the complexity in the design of a $\log_2 q \times \log_2 q$ multiplier, used once in full during the multiplication process and twice partially during the Montgomery reduction. To reduce the full complexity of the big mul-

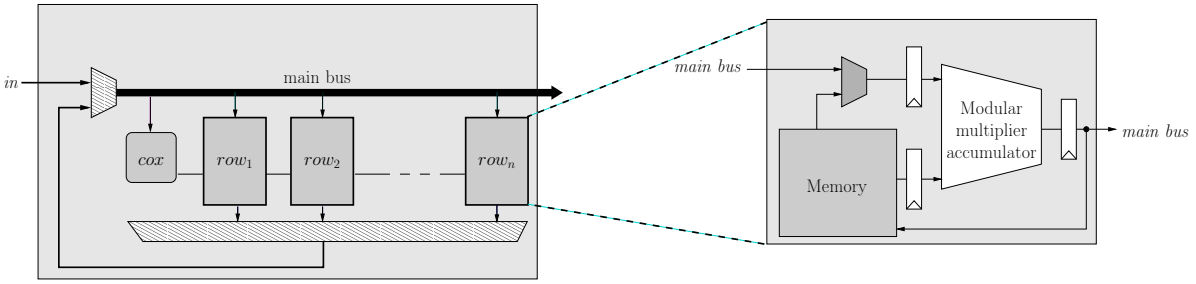

Figure 3.5: Overview of the architecture from [NMSiK01]. The values of q' and u_i are represented by $\lceil \log_2 q \rceil$ bits or more, but operations on them require a single clock cycle.

tiplication, interpolation techniques are used. In [CELL10], a classical nested Karatsuba multiplication is used, whereas [NMSiK01] relies on RNS. Both can be seen as derived from the Lagrange interpolation, with different bases.

In these architectures, a specific laser shot must cancel all the u_0 or q' at the same time to produce a null fault. To have a chance, a better solution is to use non-invasive attacks (in the sense of [JcKKP03]), such as power or clock glitches. Indeed u_0 or q' are fully manipulated on the same clock cycle (or in very few), therefore it may be more practical to make the sequencer miss an instruction instead of aiming directly at the registers.

The zero high-order bits fault attack from §3.5 is also an option. In the architecture of [CELL10], the most significant bits of u_0 can be set to 0 with a focused shot. On the other hand, the architecture of [NMSiK01] is less vulnerable to this attack, since the RNS representation makes it impractical to modify the significant bits of u_0 (see Figure 3.5).

Chapter 4

Synthesis of Fault Attacks on Cryptographic Implementations

4.1 Introduction

The main difference between the initial attack of Boneh, DeMillo and Lipton [BDL97] and those we propose in Chapter 3 lies in their level of description of RSA-CRT: whereas Boneh et al. consider a high-level algorithmic description in which modular exponentiation is treated abstractly, we consider reasonably detailed implementations, going down to algorithmic descriptions of modular multiplication. This example provides strong evidence that analyzing implementations rather than algorithmic descriptions can lead to the discovery of interesting attacks. However, it also highlights a number of difficulties with this approach:

1. the number of faulted implementations grows exponentially in the length of the original program, in particular if multiple faults are considered;

2. some fault attacks require to tamper with some (but not all) loop iterations, or to add or remove some loop iterations; hence, the number of faulted implementations cannot even be bounded solely based on the length of the original program;

3. analyzing the effects of faults becomes very involved and error-prone, in particular for programs with loops;

4. there exist multiple implementations of basic arithmetic computations, requiring to repeat the analysis for each of them;

5. there might exist numerous countermeasures against a fault attack, requiring to repeat the analysis for each of the amended implementations.

In this chapter we propose, implement, and evaluate a new approach for finding fault attacks against cryptographic implementations. Our approach is based on identifying implementation-independent mathematical properties we call *fault conditions*. We choose them so that it is possible to recover secret data purely by computing on sufficiently many data points that satisfy a fault condition. Fault conditions capture the essence of a large number of attacks from the literature, including lattice-based attacks on RSA. Moreover, they provide a basis for discovering automatically new attacks: using fault conditions, we specify the problem of finding faulted implementations as a program synthesis problem. Using a specialized form of program synthesis, we discover multiple faulted implementations on RSA that realize the fault conditions, and hence lead to fault attacks. Several of the attacks found by our tool are new, and of independent interest.

This work is to be presented at ACM CCS 2014 [BDF+14b].

Table 4.1: Fault conditions for RSA signatures. The value of ε depends on the size n of the modulus and is a multiple of the words size.

Informal description	Fault condition	Attack technique
S is a multiple of p (Prop. 4.1)	$S = 0 \bmod p \wedge S \neq 0 \bmod q$	GCD computation
S is an almost full linear combination of p and q (Prop. 4.2)	$\exists \alpha, \beta.\ S = \alpha\ p + \beta\ q \wedge \alpha, \beta < 2^{\frac{n}{2}-\varepsilon}$	Orthogonal lattices
S is an almost full affine transform of p or q (Prop. 4.3)	$\exists \alpha, \beta.\ S = \alpha\ p + \beta \wedge \alpha < q, \beta < 2^{n/2-\varepsilon}$	Orthogonal lattices

4.1.1 Our contributions

The thesis of this chapter is that it is beneficial to develop and implement rigorous methodologies for discovering fault attacks on cryptographic implementations. To support our thesis, we propose and validate experimentally a principled, tool supported approach for discovering fault attacks on cryptographic implementations. Our approach relies on two broad contributions:

1. identifying *fault conditions*, a novel concept that captures the essence of fault attacks in a logical, implementation independent setting;
2. applying a form of program synthesis on concrete cryptographic implementations to automatically discover faulted implementations that realize fault conditions and lead to attacks.

A third, more practical contribution is an evaluation of our approach on implementations of RSA. During the process, we discover several faulted implementations, some of which lead to new attacks of independent interest. We elaborate on these points below.

Fault conditions

The first contribution (Section 4.2) is of methodological nature, and rests on the introduction of *fault conditions*. Informally, fault conditions are implementation-independent mathematical properties, specific to a cryptographic system, which capture sufficient conditions under which an attacker can launch a successful attack. Consider, for instance, the case of RSA signatures with public RSA modulus $N = pq$ of length n. Any adversary with knowledge of a value S that is multiple of p but not multiple of q can obtain p by performing a simple GCD computation, and then q by division. This is captured by the fault condition

$$S : S = 0 \bmod p \wedge S \neq 0 \bmod q$$

Table 4.1 summarizes some relevant instances of fault conditions for RSA signatures. For each of the fault conditions we consider, we exhibit an attack for retrieving the secret key. Broadly speaking, the attacks fall into two categories. The first one encompasses attacks that perform an elementary computation from a value satisfying the fault condition. The second one covers attacks that require many values that satisfy the fault condition, and involve more complex computations, typically based on lattice reductions. For the latter, we implement the attacks in a computer algebra system, and we experimentally validate their effectiveness for different choices of parameters.

Fault Models and Policies

The literature offers a wide range of fault models, that affect both data flow (for example the null fault model in which integer variables can be set to a null value) and control flow (for example, the instruction skip fault model, where an instruction can be skipped). We consider various *fault policies*, that subsume a wide range of such fault model and provide fine-grained specifications of the faults that can be performed on implementations. They model faults using replacement clauses of the form (x, e) where x is a variable and e is an expression, or (c, c'), where c and c' are commands. These clauses respectively state that it is possible to replace x by e, and c by c' in the execution of the program.

Automated synthesis of faulted implementations

Identifying fault conditions that allow efficient attacks to exist is a manual process that requires cryptographic expertise, and some good understanding of the mathematical tools available for cryptanalysis. The significant pay-off of fault conditions is that the process of finding complying faulted implementations can be automated. Our second contribution (Section 4.3) is a fully automated method for discovering faulted implementations that verify the fault condition. Our method can be seen as an instance of *program synthesis*, an area that is currently undergoing rapid and significant progress (see §4.1.2). Broadly construed, the goal of program synthesis is to find, given a specification ϕ (for instance, ϕ might capture the input/output behavior of a program), a set of programs that satisfy ϕ. Because synthesis is computationally expensive, there exist many specialized forms of program synthesis that restrict the search space using non-functional requirements or by providing a partial description of the desired programs. We also specialize our synthesis algorithm to keep it computationally reasonable.

Specifically, our algorithm takes as input a fault condition ϕ, an implementation c, and searches for all faulted implementations of c that satisfy ϕ. The search is constrained by two additional inputs. The first additional input is a fault policy; the second, optional, input is an upper bound on the number of faults we allow.

Our algorithm exploits many of the standard techniques used in other approaches to program synthesis, including weakest preconditions and invariant generation, and interfaces with SMT solvers for checking the validity of first-order formulae. In addition, our algorithm relies on an automated prover to simplify the intermediate conditions generated by weakest precondition computations; the prover is specialized to formulae that combine arithmetic inequalities and size constraints; such formulae include many fault conditions, including all those we explore in this chapter (see Table 4.1). On the other hand, our algorithm noticeably departs from recent works on program synthesis by its simplicity: indeed, physical limits on the number and nature of faults sufficiently constrain the search space for faulted implementations, allowing us to dispense from using more elaborate techniques that are required to manage very large search spaces (see §4.1.2). Experimental results, which we report below, demonstrate that our synthesis algorithm performs well on standard examples.

Application: old and new attacks on RSA

The third contribution of our work is a practical evaluation of our approach on RSA signatures. We carry the evaluation using the computer algebra system SAGE, and the EasyCrypt tool. Concretely, we use the former for estimating the effectiveness of lattice-based attacks for different fault conditions, and the latter (or more precisely an implementation of our

synthesis algorithm built on top of EasyCrypt) for synthesizing faulted implementations of RSA signatures. During the process, we rediscover many known attacks; moreover, we also discover many new attacks, several of which are efficient attacks of independent interest. We summarize our main findings below:

1. For RSA–CRT signatures based on Garner's recombination, we recover the basic and most efficient attack of Chapter 3 which injects a null fault in the last call to CIOS during the computation of modular exponentiation. We also discover a new efficient attack, based on forcing additional iterations in the last call to CIOS. This attack yields almost full affine transforms of p or q, a small number of which is sufficient to recover the factorization of the RSA modulus using orthognal lattices or Simultaneous Diophantian Approximations as in [FMP03, Lag82].

2. For RSA–CRT signatures based on the usual CRT recombination, we discover a new fault attack; to our best knowledge, this is the first efficient fault attack that works with randomized padding. The attack is based on forcing additional iterations in the last call to CIOS and yields almost full linear combinations of p and q. From a small number of such faulty signatures, the factorization of the RSA modulus can easily be recovered using orthogonal lattices.

4.1.2 Related work

Formal methods for cryptography. Our work is more closely related to a recent series of articles that apply formal methods to fault attacks. However, our emphasis is on finding fault attacks against implementations, whereas other works focus on proving absence of fault attacks against algorithmic descriptions or implementations. Two independent efforts by Christofi, Chetali, Goubin and Vigilant [CCGV13] and by Rauzy and Guilley [RG13] prove the absence of fault attacks against RSA-CRT with Vigilant countermeasure. In a similar spirit, Moro et al. [MHER14] propose an approach based on redundancy to protect implementations against instruction skip attacks. The next chapter is also strongly related to formal methods for cryptography and more details about previous works will be given.

Another recent series of papers use type systems and SMT solvers for verifying whether cryptographic implementations are correctly masked [MOPT12, BRNI13, EWS14]; in particular, Eldib and Wang [EW14] have developed a method for synthesis of masking countermeasures.

Synthesis. Program synthesis is an active area of research that is undergoing rapid and significant progress, thanks to novel and practically achievable approaches, and to advances in SMT solvers. In contrast to the early works that pursue deductive program synthesis, where the program is extracted from the proof of a theorem, typically a $\forall\exists$ statement, most of the current work focuses on inductive program synthesis, and uses SMT solvers. Many works on inductive synthesis, notably early ones, have focused on loop-free programs [SLRBE05, GJTV11, JGST10]. Other recent works allow synthezing programs with loops; for instance, Srivastava, Gulwani and Foster [SGF10] introduce proof-theoretic synthesis, a variant of synthesis that combines inference of loop invariants and synthesis of loop-free programs. However, this approach is limited to programs whose loop invariants fall into a limited class of assertions. Syntax-guided program synthesis [ABJ+13] is a recently proposed framework that subsumes many of the previous approaches to synthesis. One ambition of this project is to develop a framework for testing and comparing different implementations, and in particular to provide a common input format inspired from SMT-LIB for synthesis tools. In the future, it would be interesting to suggest automated

discovery of fault attacks as a challenge for syntax-guided synthesis competitions.

Our approach shares many similarities with program repair, an instance of program synthesis that aims at automatically eliminating deficiencies in code. Informally, a program repair algorithm takes as input a program p and a property ϕ that must be satisfied by the output of p, and computes by small successive modifications of p a program p' that satisfies ϕ. There exist many approaches to program repair; some of them are based on genetic algorithms [GNFW12], others are based on code contracts [WPF+10]. We refer the reader to a recent overview [GFW13] for more information. The connection with program repair is very direct; indeed, one can even view faulted implementations as a form of program repair for the attacker. However, the techniques used in program repair are not immediately applicable to finding fault attacks on cryptographic implementations.

4.2 Fault conditions

The primary goal of fault attacks is to induce outputs which satisfy an implementation-independent, mathematical property that guarantees that the secret key or some other confidential data can be efficiently recovered. Our approach critically relies on providing a precise formalization of these mathematical conditions, using *fault conditions*. Informally, a fault condition is a statement of the form

$$v_1, \ldots, v_n : \phi \rightsquigarrow s_1, \ldots, s_k$$

where ϕ is a logical formula that depend on $v_1, \ldots, v_n$, and such that an attacker with access to sufficiently many distinct tuples of values $(v_1, \ldots, v_n)$ satisfying ϕ is able to recover secrets $s_1, \ldots, s_k$ (typically parameters of the cryptosystem) with high probability. More formally, ϕ is a first-order formula over some first-order theory T, for instance modular arithmetic, and moreover all variables that appear free in ϕ but not on the left of the colon can only denote parameters of the cryptosystem.

In this section, we introduce several fault conditions for RSA and show how, given sufficiently many satisfying values, one can efficiently retrieve either the factorization of the modulus. Many of these conditions appear implicitly in some variant form in the literature.

Convention. All the conditions we consider are of the form $v_1, \ldots, v_n : \phi \rightsquigarrow p, q$. Since the secret values $s_1, \ldots, s_k$ are determined by the case study, from now on we simply write $v_1, \ldots, v_n : \phi$.

4.2.1 Background on lattices

Lattice reduction is a powerful tool that is extensively used in the cryptanalysis of public-key cryptosystems. In this section, we provide a brief introduction to some key definitions and algorithms that are used in the chapter.

A lattice $\mathbf{L}$ is a subgroup of $\mathbb{Z}^n$, i.e. a non-empty set of vectors closed under addition and inverse. Every lattice $\mathbf{L}$ has a basis, i.e. a finite set of linearly independent vectors that generate all elements in $\mathbf{L}$. Conversely, every set $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell})$ of linearly independent vectors over $\mathbb{Z}^n$ generate a lattice $\mathbf{L} = \langle \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell} \rangle$ consisting of all integer linear combinations of the $\boldsymbol{b_i}$'s.

A central problem with lattices is to compute nearly reduced bases, i.e. bases that consist of reasonably short and almost orthogonal vectors. There exist many efficient algorithms for performing lattice reductions, including the celebrated Lenstra-Lenstra-Lovasz

(LLL) algorithm [LLL82] and Block Korkin-Zolotarev (BKZ) variants [SE94]. Lattice reduction is an essential tool in cryptanalysis, and we use it extensively in our attacks. For our purposes, it is sufficient to know that the norm of the reduced vectors for a lattice $\mathbf{L}$ of dimension ℓ output by LLL can be approximated by $\alpha\sqrt{\ell}\cdot(\det \mathbf{L})^{1/\ell}$. Hadamard's inequality allows us to upper bound $\det \mathbf{L}$ by $\prod_{i=1}^{\ell} \|\boldsymbol{b_i}\|$. In theory, LLL outputs in polynomial-time a reduced basis and each vector of the base is related to the shortest ones by an approximation factor which is exponential in the dimension. BKZ algorithms allow different tradeoff between the quality of the approximation and the time complexity. In practice, LLL implementations are very fast and when the dimension is much less than 200 [S+14], it is expected that LLL produces shorter vectors than other algorithms since its approximation factor is $\alpha \approx 1.01$, as shown experimentally in [GN08]. In larger dimensions, the approximation factor increases (unless we greatly increase the time complexity) and the success probability of our attack is reduced.

To any lattice $\mathbf{L}$ in $\mathbb{Z}^n$ is associated its *orthogonal lattice* $\mathbf{L}^{\perp}$, defined as the set of all vectors in $\mathbb{Z}^n$ that are orthogonal to all vectors of $\mathbf{L}$. The key properties of $\mathbf{L}^{\perp}$ are: $\det \mathbf{L}^{\perp} = \det \mathbf{L}$ and $\dim \mathbf{L}^{\perp} = n - \dim \mathbf{L}$. Consequently, orthogonal lattices are interesting when their dimension is very small. Indeed, when the dimension of $\mathbf{L}^{\perp}$ is higher than that of the original lattice, the bounds on the norm of the vectors in reduced bases imply that the norm of the shortest vector in $\mathbf{L}^{\perp}$ is smaller than the norm of the shortest vector in $\mathbf{L}$. It is possible to reduce the computation of the orthogonal lattice to lattice reduction in polynomial time [NS00]. Orthogonal lattices were introduced in cryptanalysis by Nguyen and Stern in [NS97], and have since found many applications [NS00].

Finding the Shortest non-zero Vector Problem of a lattice (SVP) is a well-known NP-hard problem, that is only approximated by LLL. When the dimension is small and $\alpha \approx 1$, it is likely that LLL solves this problem. In the case of orthogonal lattices, we need good properties for many reduced vectors and we assume that LLL solves this problem. In the reduced bases output by LLL, vectors are ordered according to their norm, so the last vector of the basis is also the longest. Finally, finding the closest vector of a lattice to some point (CVP) is another NP-hard problem. There exist specific algorithms to solve this problem, but in practice, we use the embedded technique, which is a heuristic reduction from the CVP to the SVP problem.

4.2.2 Fault conditions for RSA signatures

Throughout this section, we assume that N is an RSA modulus of size n, product of two large primes p and q.

Finding multiples of p or q

Our first fault condition considers faulted signatures that are a multiple of p or q. This fault condition enables attacks on RSA by simple gcd computations.

Proposition 4.1. *Given a single value S satisfying the condition:*

$$S : S \equiv 0 \bmod p \wedge S \not\equiv 0 \bmod q,$$

one can efficiently factor the RSA modulus N. Obviously the same result holds by switching p and q.

Proof. One can retrieve the factorization of N by performing a simple gcd computation between S and N.

■

This fault condition is implicit for instance in Chapter 3.

Finding "almost full" linear combinations of p and q

Our second fault condition considers faulted signatures that are linear combinations of p and q with almost full coefficients. A variant of this fault condition is implicit in [BNNT11a].

Proposition 4.2. *Assume that N is a balanced RSA modulus, i.e. p, q such that $p, q < 2^{n/2}$. Given a sufficient number of values that satisfy the fault condition:*

$$S : \exists x, y.\ S = x \cdot p + y \cdot q \wedge x, y < 2^{n/2-\varepsilon},$$

one can efficiently factor the RSA modulus N. The value ε depends on n and impacts the efficiency and success probability of the algorithm to recover the factorization.

Proof. Assume given vectors $\boldsymbol{S} = (S_1, \cdots, S_\ell)$, $\boldsymbol{x} = (x_1, \cdots, x_\ell)$ and $\boldsymbol{y} = (y_1, \cdots, y_\ell)$ such that $S_1 \dots S_\ell$ are pairwise distinct and $S_i = x_i \cdot p + y_i \cdot q$ with $x_i, y_i < 2^{n/2-\epsilon}$ for $i = 1, \dots, \ell$, so $\|\boldsymbol{x}\|, \|\boldsymbol{y}\| < \sqrt{\ell} 2^{n/2-\epsilon}$.

First, note that knowledge of $\boldsymbol{x}$ and $\boldsymbol{y}$ is sufficient to factor N since we can write a system of two linear equations using S_1 and S_2 in two variables p and q. However, it is not possible to gain knowledge of $\boldsymbol{x}$ and $\boldsymbol{y}$ directly and we will use orthogonal lattices to recover them, along similar lines as [BNNT11a]. The idea is that the orthogonal lattice of $\boldsymbol{S}$ contains $\ell - 2$ vectors that are also orthogonal to $\boldsymbol{x}$ and $\boldsymbol{y}$. More precisely, $\boldsymbol{S}^\perp = \{\boldsymbol{x}, \boldsymbol{y}\}^\perp \oplus \boldsymbol{u}$, where $\oplus$ denotes here the direct sum of lattices and $\boldsymbol{u}$ is a one dimensional lattice generated by this vector since dimension of $\boldsymbol{S}^\perp$ is $\ell - 1$. It is easy to compute the orthogonal lattice of $\boldsymbol{S}$ in polynomial-time using LLL. In the output basis, we can identify the direction $\boldsymbol{u}$ by using some conditions on the reduced vector size thanks to the structure of the vectors in $\{\boldsymbol{x}, \boldsymbol{y}\}^\perp$ that we detailed later. By removing this direction from a reduced basis of $(\boldsymbol{S})^\perp$, we get a basis of $\{\boldsymbol{x}, \boldsymbol{y}\}^\perp$. Finally, by computing the orthogonal lattice of $\{\boldsymbol{x}, \boldsymbol{y}\}^\perp$, we obtain a reduced basis $\{\boldsymbol{x'}, \boldsymbol{y'}\}$ of the lattice $\{\boldsymbol{x}, \boldsymbol{y}\}$.

Note that $\boldsymbol{x}$ and $\boldsymbol{y}$ are short vectors (we have an upper bound on them). Of course, there is no reason that they are the shortest vectors in the lattice, and $\boldsymbol{x'}$ and $\boldsymbol{y'}$ are likely different from them, so we are not done yet. However, we know that $\boldsymbol{x}$ and $\boldsymbol{y}$ are not too far from the reduced basis (we can estimate the size of the shortest vectors in $\{\boldsymbol{x}, \boldsymbol{y}\}$). Therefore, we can enumerate all short vectors in the 2-dimensional lattice $\mathbf{L'} = \{\boldsymbol{x'}, \boldsymbol{y'}\}$ to recover $\boldsymbol{x}$ and $\boldsymbol{y}$. The Gaussian heuristic allows us to estimate the number of vectors having a prescribed length and in our case, we only have to test a constant number of vectors. Indeed, $vol(\mathbf{L'}) \approx \|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\| \approx 2^{n-2\epsilon}$ and the number of lattice points of norm shorter than $d = \sqrt{\ell} 2^{n/2-\epsilon}$ in $\mathbf{L'}$ is $d^2/vol(\mathbf{L'}) \approx \ell$ since the lattice $\mathbf{L'}$ is a two-dimensional lattice.

The technical part consists in proving that we can identify $\boldsymbol{u}$ in the reduced basis of $\boldsymbol{S}^\perp$. This vector satisfies the condition

$$p \cdot \langle \boldsymbol{u}, \boldsymbol{x} \rangle + q \cdot \langle \boldsymbol{u}, \boldsymbol{y} \rangle = 0$$

since it is in $\boldsymbol{S}^\perp$, with $\langle \boldsymbol{u}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{u}, \boldsymbol{y} \rangle$ are non zero, otherwise $\boldsymbol{u}$ would also be orthogonal to $\boldsymbol{x}$ and $\boldsymbol{y}$. Consequently, the integers $\langle \boldsymbol{u}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{u}, \boldsymbol{y} \rangle$ satisfy $p \cdot a + q \cdot b = 0$ in the integer unknown a and b. Then, $(\langle \boldsymbol{u}, \boldsymbol{x} \rangle, \langle \boldsymbol{u}, \boldsymbol{y} \rangle)$ is in the orthogonal lattice of (p, q), which is a one-dimensional lattice. It is easy to see that the smallest non-zero vector is $\pm(q, -p)$ and consequently, any non-zero vector has a length larger than $2^{n/2}$, and in particular this implies that $|\langle \boldsymbol{u}, \boldsymbol{x} \rangle|, |\langle \boldsymbol{u}, \boldsymbol{y} \rangle| > 2^{n/2-1}$. This allows us to derive a lower bound on $\|\boldsymbol{u}\|$ using

Table 4.2: Minimal number of signatures ℓ to be faulted depending on the bitsize of x_i, y_i. Almost full linear combinations of p and q.

p, q	512 (bits)				1024 (bits)			
x_i, y_i	464	472	480	496	968	976	984	992
ℓ	22	26	33	74	37	44	53	67

the Cauchy-Schwarz inequality: $\|\boldsymbol{u}\| \geq 2^{\varepsilon}/(2\sqrt{\ell})$. Finally, to recover the vector $\boldsymbol{u}$ in the reduced basis of $\boldsymbol{S}^{\perp}$ which is not in $\{\boldsymbol{x}, \boldsymbol{y}\}^{\perp}$, we remove vectors which have a length larger than $2^{\varepsilon}/(2\sqrt{\ell})$. In order to have only one such vector, we force vectors of the reduced basis of $\{\boldsymbol{x}, \boldsymbol{y}\}^{\perp}$ to have length shorter than this bound, and thus $\boldsymbol{u}$ will be easily and uniquely distinguished. It is easy to estimate the size of the vectors in the lattice $\{\boldsymbol{x}, \boldsymbol{y}\}^{\perp}$ using its dimension $\ell - 2$ and volume, which can be upper bounded by Hadamard's inequality to $\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\| \approx 2^{n-2\varepsilon}$. Consequently, if $\sqrt{\ell}(2^{n-2\varepsilon})^{1/\ell-2} < 2^{\varepsilon}/(2\sqrt{\ell})$, then $\boldsymbol{u}$ will be the largest vector in the reduced basis of $\boldsymbol{S}^{\perp}$ and in practice it suffices to remove $\boldsymbol{b_{\ell-1}}$ from the reduced basis since the output vectors of LLL are in increasing order.

■

Relating this fault condition with [BNNT11a, BNNT11b] In [BNNT11a], the authors force random faults on the modulus during CRT recombination and obtain a fault condition of the following form.

$$\widehat{S} : \exists \alpha, \beta.\ \widehat{S} = \alpha \cdot p(p^{-1} \bmod q) + \beta \cdot q(q^{-1} \bmod p) \wedge \alpha, \beta < 2^{n/2}.$$

If our condition is similar to theirs, the algorithmic problem ours captures is more general. Indeed, in the analysis, the crucial parameter is the ratio between the size of p, q and the size of α, β in the relation $S = \alpha \cdot p + \beta \cdot q$. The larger this ratio is, the easier the attack is since the target vector, called $\boldsymbol{u}$ above is larger. In our case, the size of this vector is close to $2^{\varepsilon}/\sqrt{\ell}$ while [BNNT11a] consider a much larger one (their ratio is $\sqrt{N/\ell}$).

Finding "almost full" affine transforms of p or q

Our third fault condition considers faulted signatures that are almost full affine transforms of p or q. This condition is implicit in Chapter 3.

Proposition 4.3. *Assume that N is a balanced RSA modulus, i.e. p, q such that $p, q < 2^{n/2}$. Given a sufficient number of values that satisfy the fault condition:*

$$S : \exists x, y.\ S = x \cdot p + y \wedge x < q, |y| < 2^{n/2-\varepsilon},$$

one can efficiently factor the RSA modulus N. The value ε depends on n and impacts the efficiency and success probability of the algorithm to recover the factorization.

Proof. Assume given vectors $\boldsymbol{S} = (S_1, \cdots, S_\ell)$, $\boldsymbol{x} = (x_1, \cdots, x_\ell)$ and $\boldsymbol{y} = (y_1, \cdots, y_\ell)$ such that $S_1 \ldots S_\ell$ are pairwise distinct and $S_i = x_i \cdot p + y_i$ with $x_i < q$, and $|y_i| < 2^{n/2-\varepsilon}$ for $i = 1, \cdots, \ell$.

The proof is similar to the previous case, but here we are looking for the $\boldsymbol{y}$ vector since it is sufficient to recover p by computing $\gcd(N, S_1 - y_1)$ and it is the only short vector as

Table 4.3: Minimal number of signatures ℓ to be faulted depending on the bitsize of y_i. Almost full affine transforms of p or q.

p, q	512 (bits)				1024 (bits)			
y_i	464	472	480	496	968	976	984	992
ℓ	23	28	35	77	39	46	56	71

$\boldsymbol{x}$ is as large as the unknown p. Consequently, we expect to recover $\boldsymbol{y}$ as the shortest vector $\boldsymbol{x'}$ in $\{\boldsymbol{x'}, \boldsymbol{y'}\}$. The Gaussian heuristic allows us to conclude that $\boldsymbol{y}$ will be the shortest since the volume of $\mathbf{L'}$ is $\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\| < 2^{n-\varepsilon}$ using Hadamard and so $\|\boldsymbol{x'}\| \approx \sqrt{2}2^{(n-\varepsilon)/2}$. Consequently, $\|\boldsymbol{y}\| < \sqrt{\ell}2^{n/2-\varepsilon} < \|\boldsymbol{x'}\|$ for $\ell < 2^{\varepsilon+1}$.

The technical part is a little different as the condition for $\boldsymbol{u}$ is now:

$$p \cdot \langle \boldsymbol{u}, \boldsymbol{x} \rangle + \langle \boldsymbol{u}, \boldsymbol{y} \rangle = 0$$

with $\langle \boldsymbol{u}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{u}, \boldsymbol{y} \rangle$ are non zero, otherwise $\boldsymbol{u}$ would also be orthogonal to $\boldsymbol{x}$ and $\boldsymbol{y}$. Consequently, the integers $\langle \boldsymbol{u}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{u}, \boldsymbol{y} \rangle$ satisfy $p \cdot a + b = 0$ in the integer unknown a and b. Then, $(\langle \boldsymbol{u}, \boldsymbol{x} \rangle, \langle \boldsymbol{u}, \boldsymbol{y} \rangle)$ is in the orthogonal lattice of $(p, 1)$, which is a one-dimensional lattice. It is easy to see that the smallest non-zero vector is $\pm(1, -p)$ which implies $|\langle \boldsymbol{u}, \boldsymbol{y} \rangle| > 2^{n/2-1}$. As before, we can derive a lower bound on $\|\boldsymbol{u}\|$ using Cauchy-Schwarz, $\|\boldsymbol{u}\| \geq 2^{\varepsilon}/(2\sqrt{\ell})$. The end of the proof is the same except that since $\|\boldsymbol{x}\| < \sqrt{\ell}2^{n/2}$, $vol(\mathbf{L'}) < 2^{n-\epsilon}$, and so we expect to recover $\boldsymbol{u}$ provided $\sqrt{\ell}(2^{n-\varepsilon})^{1/\ell-2} < 2^{\varepsilon}/(2\sqrt{\ell})$. ■

Implementation and evaluation of key recovery

Here, we describe how the attacks outlined in the previous paragraphs can be performed. Moreover, we estimate the number of signatures required for recovering the factorization.

Implementation. We use the SAGE computer algebra system [S+14] to implement the attacks. The attacks take as input a sufficient number of signatures $S_1, \cdots, S_\ell$ satisfying the fault condition given in Proposition 4.2 or Proposition 4.3. The implementation heuristically recovers the factorization of the RSA modulus N as follows.

- Compute an LLL-reduced basis $\{\boldsymbol{b_1}, \cdots, \boldsymbol{b_{\ell-1}}\}$ of the lattice $(S_1, \cdots, S_\ell)^{\perp}$. This is done by applying LLL to the lattice in $\mathbb{Z}^{1+\ell}$ generated by the rows of the following matrix:

$$\begin{pmatrix} \kappa S_1 & 1 & & 0 \\ \vdots & & \ddots & \\ \kappa S_\ell & 0 & & 1 \end{pmatrix}$$

  where κ is a suitably large constant, and removing the first component of each resulting vector [NS97].

- Compute an LLL-reduced basis $\{\boldsymbol{x'}, \boldsymbol{y'}\}$ of the orthogonal lattice $\{\boldsymbol{b_1}, \cdots, \boldsymbol{b_{\ell-2}}\}^{\perp}$. Again, this is done by applying LLL to the lattice in $\mathbb{Z}^{\ell-2+\ell}$ generated by the rows of

$$\begin{pmatrix} \kappa' b_{1,1} & \cdots & \kappa' b_{\ell-2,1} & 1 & & 0 \\ \vdots & & \vdots & & \ddots & \\ \kappa' b_{1,\ell} & \cdots & \kappa' b_{\ell-2,1} & 0 & & 1 \end{pmatrix}$$

  and keeping the last ℓ components of each resulting vector.

- For all the linear combinations $\boldsymbol{z}$ of $\boldsymbol{x'}$ and $\boldsymbol{y'}$ that satisfy the size constraints, compute the test $\gcd(z_1 S_2 - z_2 S_1, N)$ or $\gcd(y_1 - S_1, N)$, depending on the fault condition considered, which allows to recover the prime factors p and q.

Evaluation. We use our SAGE implementation to evaluate the number of signatures required for the attacks to succeed. The results are given in Tables 4.2 and 4.3. We see for instance that 35 values are required to retrieve the factorization of N when p and q are 1024-bit and the size of the x_is and y_is have size 960, i.e. 64 bits shorter than the full size.

4.2.3 Discussion

All the fault conditions considered above are intended to predicate over the output of faulted signatures. However, fault conditions may also relate outputs of faulted and valid signatures, or inputs and outputs of signatures. Examples of such fault conditions are given by the original Bellcore attack and by Lenstra's variant:

$$\begin{aligned} S_1, S_2 &: S_1 - S_2 \equiv 0 \bmod p \wedge S_1 - S_2 \not\equiv 0 \bmod q \\ M, S &: S - M^e \equiv 0 \bmod p \wedge S - M^e \not\equiv 0 \bmod q \end{aligned}$$

Both conditions can be further refined. For instance, the fault condition for the Bellcore attack can be refined to express that one of the S_i, say for instance S_1, is a valid signature of a message m, and S_2 is a faulty signature of m. In fact, one can define a partial order on fault conditions[1] and prove that the above fault conditions are smaller than the fault condition given in Proposition 4.1.

4.3 Synthesis of Faulted Implementations

In this section, we present an automated tool that synthesizes faulted implementations that verify a fault condition. Our tool is built on top of EasyCrypt [BGHB11], a tool-assisted framework for verifying the security of cryptographic constructions.

4.3.1 Programming and assertion language

We consider programs that are written in a core imperative language with deterministic and probabilistic assignments, conditionals, loops, procedure calls, and sequential composition; the syntax of programs is given in Figure 4.1. The programming language essentially

[1] $(v_1, \ldots, v_n : \phi) \leq (w_1, \ldots, w_m : \psi)$ if there exists an efficient and public n-ary function f that returns m-tuples of values and such that for every $v_1, \ldots, v_n$ such that $\phi(v_1, \ldots, v_n)$ holds and for every $w_1, \ldots, w_m$ such that $f(v_1, \ldots, v_n) = (w_1, \ldots, w_m)$, we also have $\psi(w_1, \ldots, w_m)$.

$$
\begin{array}{lll}
\mathcal{C} ::= & \mathsf{skip} & \\
\mid & \mathcal{C}; \mathcal{C} & \text{sequencing} \\
\mid & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\
\mid & \mathcal{V} \xleftarrow{\$} \mathcal{DE} & \text{random assignment} \\
\mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
\mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\
\mid & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mid & \textbf{return}\ \mathcal{E} & \text{return expression}
\end{array}
$$

where $\mathcal{V}$ denotes the set of *variables* and $\mathcal{E}$ denotes the set of *expressions* and $\mathcal{DE}$ denotes the set of distribution expressions and $\mathcal{P}$ denotes the set of procedures.

Figure 4.1: Syntax of programs

subsumes the language proposed in [BR06] and in particular is sufficiently expressive to capture cryptographic implementations.

Expressions used in programs, for instance on the right-hand side of assignments or as guards in conditional statements and loops are built inductively from user-defined constants, operators, and variables. In this chapter, we specifically focus on expressions that are built from operations for modular arithmetic.We use a simple type system for expressions and programs, and we only consider well-typed programs.

Assertions are first-order formulae over the theories inherited from the expression language. Reasoning about assertions is delegated to the EasyCrypt proof engine, which can either use lemmas from libraries or invoke SMT solvers to prove the validity of an assertion.

4.3.2 Fault models and fault policies

Fault models

Fault models are high-level specifications of the type of faults that can be injected on embedded devices; they generally target specific architectures, and are designed to reflect the effects and capacities of specific perturbation techniques.

For the purpose of this chapter, it is sufficient to know that there exist two broad classes of fault models. The first class captures faults that modify the dataflow, for instance by setting a particular register to a default value (the null fault model) or to a constant but unknown value (the constant fault model), or by setting part of the register to a constant value (the zero high-order bits fault model and its variants). In practice, it is often important to consider models that combine several kinds of faults; for instance, one can consider a fault model which allows null faults on small registers, and constant faults on larger registers. Such faults are considered for example in [FLV12], where the authors also justify their practical feasibility. The second class captures faults that modify the control flow, for instance by skipping an instruction (the instruction skip model), by forcing a conditional instruction to enter into a specific branch (the branch fault model), or by forcing the execution of a loop to be interrupted before the guard is set to false, or continued after it is set to false (the loop fault model). These models are classic and are considered in [PV04], for instance. Both models overlap, in the sense that one can sometimes achieve the same effect by a dataflow fault attack, or by a control flow fault

attack.

Fault policies

Instead of hardcoding the different fault models, our tool allows users to specify fine-grained fault policies that delineate very precisely the space of faulted implementations by describing which faults can be injected in the program. Fault policies are program specific, and are given by two sets of replacement clauses.

The first set consists of variable replacement clauses of the form (x, e) where x is a program variable and e is an expression; such a clause says that one can replace the variable x by the expression e in the course of program execution. These declarations can be used to model data faults; for instance, the null fault on x is captured by the clause $(x, 0)$, whereas the zero high order bits fault on x that sets r bits to zero is captured by the clause $(x, \text{MSB}_r(x))$.

The second set consists of command replacement clauses of the form (c, c'), where c and c' are commands; such a clause says that one can replace the command c by the command c' in the course of program execution. These declarations can be used to model control flow faults; for instance, instruction skip faults on an assignment c is captured by the clause (c, skip), whereas branch faults are captured by the clause $(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, c_i)$ where $i = 1$ if the goal is to force execution to go into the true branch, and $i = 2$, otherwise. By convention, we require that all instruction replacements do not increase the set of modified variables, i.e. the set of modified variables of a command c' is a subset of the set of modified variables of the command c it replaces. This is the case for all control flow attacks described above, and is essential for the completeness of our tool.

Although it is useful in practice, fault policies do not currently include a mechanism to impose any locality constraint on the clauses, i.e. replacements may occur anywhere in the program. This can easily be circumvented by writing programs in pseudo-SSA form, for instance by adding subscripts for the different occurences of the same variable in the program.

Finally, fault policies may also include some upper bounds on the number of times a clause can be used to fault an implementation. This is useful to constrain the space of faulted implementations and to match physical constraints.

Discussion

There is a direct relation between fault models and fault policies, in the sense that every fault model determines a unique fault policy for each program. However, many fault attacks require multiple faults and can only be captured by hybrid fault models, that combine several simpler ones. An example of hybrid fault model is one that considers null faults on variables that denote small registers (for instance, variables that store values smaller than 2^8), and constant faults on variables representing larger registers.

It would be interesting to develop a high-level language for describing hybrid fault models, and a compiler for generating automatically fault policies from high-level specifications. However, building the compiler requires a significant amount of infrastructure, including the ability to automatically infer program invariants: for the example discussed above, the compiler would need to infer that the value held by a variable x is always smaller than 2^8 in order to generate the clause $(x, 0)$. We leave the design of this high-level language and the implementation of the compiler for future work, and require for now that fault policies (albeit in some edulcorated form) are given as input to the tool.

4.3.3 Algorithm

Our tool takes as input a (non-faulted) implementation written in the programming language of EasyCrypt, a fault condition, and a fault model, and optionally a precondition ψ. It outputs a set of faulted implementations that satisfy the fault condition and are valid faults of the original implementation with respect to the fault model considered. The core of the tool is an algorithm that interleaves the computation of weakest preconditions, logical simplifications, and generation of faults. For simplicity, we describe a non-deterministic and inefficient version of the algorithm, whereas the implementation uses a more efficient implementation, and some caching and early pruning techniques for the smart exploration of the search space. We initially explain how the algorithm works on straightline programs, i.e. programs without loops, conditionals, and procedure calls. Then, we explain how to extend the algorithm to procedure calls and loops. First, we define the notion of faulted instruction.

Faulted commands

The fault policy determines for each command c of the program a set of faulted instances, consisting of commands c' that can be obtained from c according to the fault policy. All commands are faulted instances of themselves, and moreover the command c' is a faulted instance of c if there exists an instruction replacement clause (c, c'). Moreover, there are some specific rules for each construct of the language.

- $x \leftarrow e[e_1, \ldots, e_n/y_1, \ldots, y_n]$ is a faulted instance of $x \leftarrow e$, provided for $i = 1 \ldots n$, the replacements of y_i by e_i are allowed by the fault policy.
- the commands $\mathsf{while}\ b\ \mathsf{do}\ c'$, and $\mathsf{if}\ b\ \mathsf{then}\ c'; \mathsf{while}\ b\ \mathsf{do}\ c$, and $\mathsf{while}\ b'\ \mathsf{do}\ c; \mathsf{if}\ b\ \mathsf{then}\ c'$ are all faulted instances of $\mathsf{while}\ b\ \mathsf{do}\ c$, where c' is a faulted instance of c, and b' is a guard that forces exactly one less iteration of the loop body.

The last clause captures faults on the first and last iteration of a loop, and can be extended to model faults on the first and last k iterations of a loop, for $k \geq 1$.

Straightline programs

The algorithm is given as input a fault policy, and manipulates triples of the form $(c, \phi, \widehat{c})$. Initially, the algorithm is given the triple (c, ϕ, skip) consisting of the program being analyzed against fault attacks, the fault condition, and the empty statement. At each iteration, the algorithm consumes the last command of c and outputs a new triple $(c', \phi', \widehat{c'})$ as follows;

1. c is decomposed into a sequence $c'; i$, where i is the last command of the program (necessarily an assignment or a random sampling). If c is empty, then the algorithm checks if ϕ is a consequence of the precondition, and returns $\widehat{c}$ if this is the case and nothing otherwise;
2. the algorithm checks whether i affects ϕ, i.e. if any of the variables modified by i occur in ϕ. If not, the algorithm breaks to the next iteration with $(c', \phi, \widehat{c'})$, where $\widehat{c'} = i; \widehat{c}$;
3. if some variable modified by i occurs in ϕ, then the algorithm chooses non-deterministically a faulted instance i' of i;

4. the algorithm computes the weakest precondition of i' on ϕ. For instance, the rules for computing weakest preconditions of deterministic and random assignments are:

$$\begin{aligned} \mathcal{WP}(x \leftarrow e, \phi) &= \phi\{x \leftarrow e\} \\ \mathcal{WP}(x \xleftarrow{\$} d, \phi) &= \forall v \in \mathsf{dom}(d), \phi\{x \leftarrow v\}, \end{aligned}$$

where $\mathsf{dom}(d)$ is the set of values that have a non-zero probability in d. Note that the weakest precondition computation takes an assertion and returns an assertion. This is achieved by viewing probabilistic assignments as non-deterministic assignments over the domain of the distribution from which the assignment is sampled;

5. the algorithm applies logical simplifications to the assertion ϕ output by the weakest precondition computation. The output is a new assertion ϕ' that has fewer free variables than ϕ;

6. the algorithm proceeds to the next iteration with $(c, \phi', \widehat{c'})$, where $\widehat{c'} = i'; \widehat{c}$.

Breaking to the next iteration in step 2 and performing logical simplifications in Step 5 may in fact significantly prune the search space, without ruling out any potential attacks: computing the weakest precondition on a command i whose left-hand-side does not appear in the fault condition never changes that fault condition, whichever fault may be selected. Indeed, our algorithm is sound and relatively complete for straightline code, in the sense that, given an oracle that can decide logical implications, the algorithm would return all faulted versions c' of c such that the Hoare triple $\{\psi\}c'\{\phi\}$ is valid. In practice, logical implications are verified using SMT solvers, and hence the implementation might actually fail to find a valid fault attack.

Procedure calls

Our tool deals with programs that make non-recursive procedure calls by entering into the code of the procedure when reaching a call. This is intuitively equivalent to inlining all procedure calls and applying the non-procedural analysis to the inlined code. Although it is certainly possible to develop more sophisticated approaches, including ones that deal with recursive procedure calls, based on state-of-the-art techniques, our elementary approach has the advantage of simplicity and is sufficient for most implementations of cryptographic algorithms.

Loops

Dealing with loops is the main source of complexity for our tool, as computing weakest preconditions requires knowing some useful loop invariants, i.e. assertions that hold throughout all iterations of the loop body. We provide two elementary mechanisms for dealing with loops: an invariant generator, and an algorithm for turning (user-provided) invariants for non-faulted loops into invariants for their faulted instances. There is admittedly significant scope for improving these mechanisms, in particular by building upon recent developments in invariant generation; we leave this avenue for future work.

Pruning

We use two main pruning techniques for improving the efficiency of the search algorithm. First, since SMT solvers are a clear performance bottleneck, we cache all SMT queries and

their result. Second, we maintain a table of all intermediate statements $(c, \phi, \widehat{c})$, and abort execution whenever the algorithm computes a triple which coincides in the first and second component with an element of the table.

4.4 Applications on RSA–CRT signatures

Using our tool, we are able to discover many attacks on implementations of RSA–CRT signatures. Several of these attacks are new, and of independent interest. In this section, we review in some detail the most relevant attacks we find.

We consider a CRT-based implementation of RSA that uses the Montgomery ladder (Figure 3.1) for modular exponentiation and the CIOS algorithm (Algorithm 3.1) for modular multiplication. We consider implementations using both Garner's recombination algorithm and the standard CRT recombination with optimizations. Most of the attacks we find involve faults injected during the last call to CIOS in the ladder (line 13, Figure 3.1), which takes the result of the exponentiation back into its classical representation. We assume that the parameter x in CIOS is stored in a shift-register, used to extract its individual digits in base b.

Finding multiples of p or q

Using the fault condition from Proposition 4.1, and allowing null faults on small variables (that contain integers mod b) we recover the most basic and efficient attack of Chapter 3, which sets q' to 0 during the final call to $\mathsf{CIOS}(\bar{S}_q, 1)$. In addition, the tool also finds several variants of the fault, indicating which (combinations of) variables can be set to 0 to fulfill the fault condition. For example, setting both u_j and x_j to 0 throughout the computation still yields a null result.

This attack and its variants only work when the final call to CIOS occurs with 1 as second argument. This is not always the case when CRT recombination is used, since the call to CIOS can be used to optimize a multiplication away. In this case, by adding control flow faults to the fault policy, our tool also finds that faulting q' to 0 and doubling the number of loop iterations during this final call forces its result to zero. Indeed, in this case, after the normal number of iterations, the shift-register initially containing x now contains zero and any further loop iteration simply shifts a to the right, eventually forcing it to zero as well. A much simpler, albeit much less elegant, control flow fault involves simply faulting the initial loop condition so no computation is performed.

Finding "almost full" linear transforms of p or q

It may not always be possible to skip the loop entirely, or to ensure that the loop is run at least twice as many times as expected. However, it may be easier to inject faults on loop counters that consistently add a small (possibly unknown) number of iterations. Our tool automatically finds that such faults, when q' is set to zero during the additional loop iterations are in fact sufficient to guarantee the fault condition from Proposition 4.3 using both Garner and CRT recombination. For each additional iteration, the size of the exponentiation's result is reduced by the size of a base b digit, quickly leading to a result that can be exploited by the classic lattice-based attacks described in Section 4.2.2.

Alternatively, instead of faulting the control flow and a variable, our tool also finds that simply setting q' and x_j to zero during the last iterations of the loop leads to a similarly faulted signature, that fulfills the desired fault condition.

Finding "almost full" linear combinations of p and q

When given the fault condition from Proposition 4.2, our tool finds that running the previous size-reducing attacks on both half-exponentiations yields a suitable faulted signature when using the classic CRT recombination rather than Garner's. The relative efficiency of the lattice-based attack from Section 4.2.2 compared to the one from Section 4.2.2 may justify the additional faults.

4.5 Concluding remarks

We have presented a new approach to discover automatically fault attacks on cryptographic implementations. The technical core of our approach is a new and practical form of program synthesis. Pleasingly, the tool that implements our approach is able to discover new and interesting attacks. An exciting perspective for further work is to apply our tool to an extensive class of implementations. There are also interesting directions for improving and extending our tool. The first one is to integrate state-of-the-art invariant generation and synthesis techniques in the tool. Another one is to implement a synthesis algorithm based on relational verification in order to deal with relational fault conditions, i.e. fault conditions that relate faulted and valid signatures. Although cast in a different context, the work reported in [BCG+13] provides an excellent starting point. Yet another one would be to use synthesis for discovering countermeasures against fault attacks as done in [ÁH14] for side-channel attacks.

Chapter 5
Making RSA–PSS Provably Secure Against Non-Random Faults

5.1 Introduction

In this chapter, we prove the security of an infective countermeasure against a large class of non-random faults; the proof extends Coron and Mandal's result to a strong model where the adversary can choose the value of the faulty signatures modulo one of the prime factors of the RSA modulus. This fault model is clearly strictly more general than Coron and Mandal's. Such non-random faults induce, together with the infective countermeasure, more complex probability distributions than in the original proof; we analyze them using careful estimates of character sums over finite fields. The security proof is formally verified using appropriate extensions of EasyCrypt, and provides the first application of formal verification to provable (i.e. reductionist) security in the context of fault attacks.

This work was presented at CHES 2014 [BDF+14a].

5.1.1 Infective countermeasures

Checking results before release is a simple and practical security measure, but it is not sufficient by itself, since simple tests can be easily bypassed by flipping the outcome of a comparison [AK97, TK10]. Infective countermeasures are an alternate approach in which results are released all the time, but become gibberish when faulty computations occur: a fault (usually not controlled by the adversary) results in a random value, which consequently makes the faulty signature random. From a security point of view, since faults may not be random, we may not be able to prove that the faulty output is fully random. However, one may ask that the output be independent of secret information even in the presence of non-random faults.

Infective countermeasures have been used before by Canetti and Goldwasser [CG99] to deal with fault-injecting adversaries when decrypting ciphertexts in a distributed manner. One such countermeasure for RSA–CRT was proposed by Boscher, Handschuh and Trichina [BHT10]. In their technique, the signer computes the signature S and recomputes $y' = S^e \bmod N$ to check the signature against the padded message y, before returning $S + y'_p - (y \bmod p) + y'_q - (y \bmod q)$ if $y' = y$, and an error otherwise. Even if the adversary bypasses the verification $y' = y$, the output signature mixes the fault and correct signature in a non-trivial way. Still, this countermeasure was later attacked by Trichina and Korkikyan [TK10] for deterministic padding schemes.

We tackle the problem of masking faulty signatures so as to prevent the exploitation of faults and protect validity checks.

5.1.2 Our contributions

In this chapter we generalize the fault model from [CM09] and consider a very powerful adversary able to inject *non-random* faults. More precisely, we let the adversary set the value modulo q of the computed signatures to an arbitrary value of his choice. Clearly, since he could choose that value randomly, the model is strictly more powerful than the one considered by Coron and Mandal. In addition, it captures many other types of faults, such as the "null faults" and "constant faults" which consist in putting the result of a computation to zero or to a constant. Note that contrary to Chapter 3 where they are applied to a small register, they reflect in this chapter the result of the whole exponentiation modulo q. If such a signature is directly returned to the adversary, he can clearly factor the modulus, but we consider a simple countermeasure to avoid that problem. The countermeasure, described in Algorithm 5.1, uses infective techniques, mixing additional randomness into faulty signatures in a *provably secure way*.

In practice, we show that our random infection masks faulty signatures enough for us to prove the security of RSA–PSS under the RSA assumption in the random oracle model if enough additional randomness is provided.

Concretely, we sample a random value r' and add $r' \cdot (y - y')$ to the signature mod N, where y is the original padded message and y' is the padded message recovered from the signature. When the signature is computed correctly, $(y - y')$ is zero and the correct signature is returned. If the signature is faulty, we show that the masked output is statistically close to uniform and hence leaks no secret information.

We prove such results in two key lemmas corresponding to [CM09, Lemmas 1, 2]. Since our faults are non-random, the probability distributions are more complex; we use careful estimates of exponential sums attached to corresponding rational functions to establish their regularity. We only analyze this countermeasure when the validity check is performed in the standard way (by computing the public permutation), but our random infection might also be used to protect other checks such as Rivain's [Riv09, LRT14].

In the same way, although we use RSA–CRT as a motivating example, our fault model is in fact independent of the way the modular exponentiation is implemented, and is not limited to fault attacks on RSA–CRT.

Algorithm 5.1 Protected signing algorithm.

function SIGN(sk, pk, m) ▷ $sk = (d_p, d_q, \alpha_p, \alpha_q, N)$, $pk = (e, N)$
  $r \leftarrow \{0,1\}^{k_0}$ ▷ Start of PSS padding
  $\omega \leftarrow \mathcal{H}(m, r)$
  $st \leftarrow \mathcal{G}(\omega) \oplus (r \,||\, 0^{k_g - k_0})$
  $y \leftarrow \mathsf{os2ip}(0 \,||\, \omega \,||\, st)$
  $S_p \leftarrow y^{d_p} \bmod p$ ▷ Signature computation
  $S_q \leftarrow y^{d_q} \bmod q$
  $S \leftarrow (\alpha_p \cdot S_p + \alpha_q \cdot S_q) \bmod N$ ▷ $\alpha_p = q \cdot (q^{-1} \bmod p)$ and similarly for α_q
  $y' \leftarrow S^e \bmod N$
  $r' \leftarrow \{0,1\}^{\rho} \backslash \{0\}$ ▷ Infective countermeasure
  $S' \leftarrow S + r' \cdot (y - y') \bmod N$
  return $\mathsf{i2osp}(S')$
end function

The second contribution of the chapter is a formal proof of security of the countermeasure using EasyCrypt[1], a computer-aided framework that has previously been used to reason

[1] https://www.easycrypt.info

about the security of cryptographic constructions—but was never applied to fault attacks and countermeasures. Our proof is the first application of formal verification to provable security against fault attacks, as other works [CCGV13, MHER14, RG13] applying formal verification to fault attacks are focused on proving the correctness of the countermeasures (that is, that the protected program either returns the same result as the original program, or fails), but do not provide any provable security guarantees. Apart from increasing our confidence in the effectiveness of the countermeasure, our formal proof reveals a glitch in the proof of Coron and Mandal [CM09], and also paves the way for formally verifying the effectiveness of the countermeasures on standard implementations of PKCS probabilistic signing, in the same way that [ABBD13] uses an older prototype of EasyCrypt [BGHB11] to prove security of an implementation of PKCS encryption.

5.1.3 Related work

Christofi et al. [CCGV13] use a combination of program transformation and verification techniques for proving Vigilant's countermeasure for CRT-RSA. They take a source program p and output a program $\hat{\mathsf{p}}$ that contains all possible faulty behaviors of p. Then, they show that the program $\hat{\mathsf{p}}$ either returns a value that matches the value returned by p on the same input, or else returns an error, they conclude that the program is correct for all faults. While it is a natural guarantee to seek, their theorem does not constitute a proof of security in the sense of provable security, but rather a heuristic to validate a countermeasure implementation.

Rauzy and Guilley [RG13] develop symbolic methods to analyze fault attacks against RSA–CRT implementations. They model arithmetic computations as algebraic expressions, and define a simplification procedure for expressions. Given an expression e (representing the algorithm to be attacked), their tool tests for all possible faulty variants $\hat{\mathsf{e}}$ of e if the expression $\gcd(N, \mathsf{e} - \hat{\mathsf{e}})$ simplifies to a prime factor of the RSA modulus. If some expression $\hat{\mathsf{e}}$ is found, then the algorithm is considered insecure. Their tool is useful to find fault attacks on an algorithm, but only provides guarantees of security against a restricted class of attackers. Moreover, it is specialized to deterministic signature schemes and cannot deal with randomized paddings like PSS.

Moro et al. [MHER14] focus on the specific class of instruction skip attacks, in which an adversary forces to skip the execution of a targeted instruction. To protect against skip attacks, they transform a program p into a fault-tolerant program $\hat{\mathsf{p}}$, by providing for each instruction a possible replacement for execution in the presence of instruction skip faults. Using a model checker, they establish the equivalence between executing the instruction without faults and executing the replacement sequence of instructions with instruction skip faults. Their approach is general, and significantly improves resistance against instruction skip attacks. However, it is not suitable for obtaining the strong guarantees required by provable security.

5.2 Our results

Instead of considering the many possible faults an adversary could inject in Figure 5.1, we give the adversary access to two distinct oracles (Figure 5.1) that compute valid signatures (oracle $\mathcal{S}$) and generalize faulty signatures (oracle $\mathcal{F}$), as justified in §5.2.

As discussed, our fault model is independent of the algorithm used to compute modular exponentiation. We therefore keep simpler definitions for public and secret key: i.e. $pk = (e, N)$ and $sk = (d, p, q)$. Throughout the security proof, we assume that the modulus is

oracle $\mathcal{S}(m)$
- $r \leftarrow \{0,1\}^{k_0}$
- $\omega \leftarrow \mathcal{H}(m, r)$
- $st \leftarrow \mathcal{G}(\omega) \oplus (r \,||\, 0^{k_g - k_0})$
- $y \leftarrow \mathsf{os2ip}(0 \,||\, \omega \,||\, st)$
- $S \leftarrow y^d \bmod N$
- **return** $\mathsf{i2osp}(S)$

oracle $\mathcal{V}(m, S)$
- $r \leftarrow \perp$
- $s \leftarrow \mathsf{os2ip}(S)$
- **if** $0 < s < N$ **then**
  - $y \leftarrow s^e \bmod N$
  - $b \,||\, \omega \,||\, st \leftarrow \mathsf{i2osp}(y)$
  - $r \,||\, \gamma \leftarrow st \oplus \mathcal{G}(\omega)$
  - $\omega' \leftarrow \mathcal{H}(m, r)$
  - $r = b = 0 \wedge \omega = \omega' \wedge \gamma = 0^{k_g - k_0}$
- **end if**
- **return** r

oracle $\mathcal{F}(m, a)$
- $r \leftarrow \{0,1\}^{k_0}$
- $\omega \leftarrow \mathcal{H}(m, r)$
- $st \leftarrow \mathcal{G}(\omega) \oplus (r \,||\, 0^{k_g - k_0})$
- $y \leftarrow \mathsf{os2ip}(0 \,||\, \omega \,||\, st)$
- $S \leftarrow y^d \bmod N$
- $r' \leftarrow \{0,1\}^{\rho} \backslash \{0\}$
- $S' \leftarrow y^d \cdot \alpha_p + (a + r' \cdot (y - a^e)) \cdot \alpha_q$
- **return** $\mathsf{i2osp}(S')$

Figure 5.1: Oracles in our fault model

balanced, that is $N = p \cdot q$ is such that $2^{n-1} \leq N < 2^n$ and $2^{n/2-1} \leq p < q < 2^{n/2}$.

PSS padding is computed using two hash functions $\mathcal{H}$, outputting bitstrings of length k_h, and $\mathcal{G}$, producing bitstrings of length k_g, where $k_h + k_g + 1 = n$. In addition, the padding scheme uses a random salt of length $k_0 < k_g$.

For simplicity, we model $\mathcal{H}$ as a function from $\{0,1\}^* \times \{0,1\}^{k_0}$ to $\{0,1\}^{k_h}$, and $\mathcal{G}$ as a function from $\{0,1\}^{k_h}$ to $\{0,1\}^{k_g}$. This is done without loss of generality.

In algorithm and game descriptions, we denote with $\mathsf{i2osp}$ and $\mathsf{os2ip}$ the conversions between integers and their binary representations. For simplicity, $\mathsf{i2osp}$ always produces a bitstring of length n.

Under the Generalized Riemann Hypothesis, we reduce the $\mathcal{UF}$-$\mathcal{CMA}$ security of the faulty signature scheme presented in Fig. 5.1, where the adversary is given access to the faulty signature oracle along with the valid signature oracle and the random oracles $\mathcal{H}$ and $\mathcal{G}$, to the one-way security of RSA. We consider a forgery valid even if it was produced by the faulty signature oracle. In the rest of this chapter, we use $\mathcal{S}$ to denote the valid signature oracle, $\mathcal{F}$ to denote the faulty signature oracle, $\mathcal{K}$ to denote the RSA key generation algorithm, and $\mathcal{V}$ for the PSS verification algorithm. Subscripts identify the game in which a particular oracle appears. We denote with $\mathcal{Q}^{\mathcal{X}}$ the set of query-response pairs for queries made to oracle $\mathcal{X}$ so far.

game $\mathcal{UF}$-$\mathcal{CMA}$
- $(e, d, N) \leftarrow \mathcal{K}()$
- $(m, s) \leftarrow \mathcal{A}^{\mathcal{S},\mathcal{F},\mathcal{H},\mathcal{G}}(e, N)$
- $b \leftarrow \mathcal{V}(m, s)$
- $win \leftarrow b \wedge (m, s) \notin \mathcal{Q}^{\mathcal{S}}$
- **return** win

game $\mathcal{OW}$-$\mathcal{RSA}$
- $(e, d, N) \leftarrow \mathcal{K}()$
- $x^* \leftarrow [0..N)$
- $y^* \leftarrow x^{*e} \bmod N$
- $x \leftarrow \mathcal{I}(e, N, y^*)$
- **return** $x = x^*$

Figure 5.2: Initial and Final Games

Theorem 5.1 ($\mathcal{UF}$-$\mathcal{CMA}$ security of protected PSS in the presence of faults)**.** *Assume that the Generalized Riemann Hypothesis holds. For all $\delta > 0$, there exists a constant $\kappa_\delta > 0$ depending only on δ such that given a CMA adversary $\mathcal{A}$ against the faulty signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{F}, \mathcal{V})$ that makes at most $q_\mathcal{H}$ queries to $\mathcal{H}$, $q_\mathcal{G}$ queries to $\mathcal{G}$, $q_\mathcal{S}$ queries to $\mathcal{S}$ and $q_\mathcal{F}$ queries to $\mathcal{F}$, we build a one-way inverter $\mathcal{I}$ such that*

$$\Pr[\mathcal{UF}\text{-}\mathcal{CMA} : win] \leq \Pr[\mathcal{OW}\text{-}\mathcal{RSA} : x = x^*] + \epsilon_0$$

with

$$\epsilon_0 = \frac{(q_\mathcal{H} + q_\mathcal{S} + q_\mathcal{F}) \cdot (q_\mathcal{H} + q_\mathcal{G} + q_\mathcal{S} + q_\mathcal{F}) + q_\mathcal{G} \cdot q_\mathcal{F} \cdot 3 + 1}{2^{k_h}} + \frac{q_\mathcal{G} \cdot q_\mathcal{F}}{2^{k_h/2}}$$
$$\frac{(q_\mathcal{S} + q_\mathcal{F}) \cdot (2 \cdot q_\mathcal{H} + q_\mathcal{S} + q_\mathcal{F}) + q_\mathcal{H} + q_\mathcal{S}}{2^{k_0}} + \frac{1}{2^{\frac{k}{2}-1}} + q_\mathcal{F} \cdot 2\kappa_\delta \cdot 2^{\frac{n\delta-\rho}{2}}$$

Remark. The constant κ_δ is as in Lemma 5.1. As observed in Remark 5.3, for large enough N, it suffices to take ρ slightly larger than a given ϵ to bound the final term by $2^{-\epsilon}$. In addition, as mentioned in Remark 5.3, we assume that ρ is chosen slightly larger than k_h so that the assumptions of Lemma 5.3 are satisfied.

Fault model justification

Our faulty signature oracle computes the correct padded message y, samples r' and returns $S' = y^d \cdot \alpha_p + (a + r'(y - a^e)) \cdot \alpha_q$ with $a \in \mathbb{Z}$ chosen by the adversary.

We allow multiple faults to be injected, but only during the RSA–CRT computation (lines 6-7 of the protected signing Fig. 5.1). More precisely, we consider a scenario where the computation modulo p is correct whereas those modulo q is faulted to result in a constant a chosen by the adversary, *i.e.* $S_f = (y^d \bmod p, a) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}$. Then, using our countermeasure we obtain:

$$\begin{aligned} S' &= S_f + r'(y - S_f^e) \\ &= y^d\alpha_p + \alpha_q a + r'(y - (y^d\alpha_p + \alpha_q a)^e) \\ &= y^d\alpha_p + (a + r'(y - a^e))\alpha_q. \end{aligned}$$

Our fault model leverages the results of Coron and Mandal in [CM09] who treated the case of random faults against PSS scheme and some of those described in Chapter 3. Indeed we showed that "null faults" on a small register result in putting the signature modulo q to zero. Moreover, we take into account all the highly non-random faults or combinations of faults leading to such result, i.e. $S_q = a$ with a a constant.

5.3 Statistical Lemmas

We need several results on the regularity of the probability distributions related to the infective countermeasure. Recall that the *statistical distance* between a random variable X on a finite set S and the uniform distribution is defined as:

$$\Delta_1(X) = \frac{1}{2} \cdot \sum_{s \in S} \left| \Pr[X = s] - \frac{1}{|S|} \right|.$$

We say that X is *δ-statistically close to uniform* when $\Delta_1(X) \leq \delta$.

Our proofs rely on Dirichlet characters and character sums over $\mathbb{Z}/q\mathbb{Z}$. The basic definition of a character is as follows.

Definition 5.1. *Let q be a positive integer. A Dirichlet character modulo q is an arithmetic function χ with the following properties:*

- *χ is periodic modulo q, i.e. $\chi(n+q) = \chi(n)$ for all $n \in \mathbb{N}$.*
- *χ is completely multiplicative, i.e. $\chi(nm) = \chi(n)\chi(m)$ for all $n, m \in \mathbb{N}$ and $\chi(1) = 1$.*
- *If $(n, q) \neq 1$ then $\chi(n) = 0$, otherwise $\chi(n) = e^{2\pi i\nu/\phi(q)}$, for some $\nu \in \mathbb{N}$, with $\phi(q)$ the Euler's totient.*

We denote by $\overline{\chi}$ the inverse of a character χ with $\overline{\chi}(n) = \overline{\chi(n)}$ and χ_0 the principal character modulo q defined by $\chi_0(n) = 1$ if $(n, q) = 1$ and 0 otherwise. The following theorem, which establishes the orthogonality relations for Dirichlet characters, will be useful for the next proof.

Theorem 5.2. *Let q be a positive integer. For any integers $n_1, n_2 \in \mathbb{N}$ we have*

$$\sum_{\chi \bmod q} \chi(n_1) = \begin{cases} \phi(q) & \text{if } n_1 = 1 \bmod q \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{\chi \bmod q} \chi(n_1)\overline{\chi(n_2)} = \begin{cases} \phi(q) & \text{if } n_1 = n_2 \bmod q \quad \text{and } (n_1, q) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where the summation runs over all Dirichlet characters modulo q.

The main statistical result can be stated as follows.

Lemma 5.1. *Consider integer intervals $\mathcal{X} = [1, X]$, $\mathcal{W} = [w_0, w_0+W)$ whose lengths X, W satisfy $X, W < q$, and for all $t \in \mathbb{Z}/q\mathbb{Z}$, denote by $T(\mathcal{X}, \mathcal{W}; t) = \frac{XW}{q} \cdot \big(1 + V(\mathcal{X}, \mathcal{W}; t)\big)$ the number of solutions $(x, w) \in \mathcal{X} \times \mathcal{W}$ of the congruence $xw \equiv t \pmod q$. Assuming that the Generalized Riemann Hypothesis holds, then for all $\delta > 0$, there exists a constant $\kappa_\delta > 0$ depending only on δ (and not $q, \mathcal{X}, \mathcal{W}$) such that:*

$$\sum_{t \in \mathbb{Z}/q\mathbb{Z}} \big|V(\mathcal{X}, \mathcal{W}; t)\big| \leq \frac{\kappa_\delta q^{3/2+\delta}}{\sqrt{XW}}.$$

In particular, the distribution of the products $xw \bmod q$ is statistically close to uniform in $\mathbb{Z}/q\mathbb{Z}$ whenever $XW \gg q^{1+3\delta}$.

Proof. Note first that all elements of $\mathcal{X}$ are invertible modulo q, whereas at most one element of $\mathcal{W}$ is divisible by q. Denote by W^* the number of elements of $\mathcal{W}$ which are invertible modulo q, which is thus equal to W or $W - 1$. We then have:

$$T(\mathcal{X}, \mathcal{W}; 0) = X \cdot (W - W^*) \leq X \quad \text{and hence} \quad \big|V(\mathcal{X}, \mathcal{W}; 0)\big| \leq \frac{q}{W}.$$

On the other hand, for $t \neq 0$, we can express $T(\mathcal{X}, \mathcal{W}; t)$ as a sum over the multiplicative characters modulo q. Indeed, the orthogonality of characters (Theorem 5.2)ensures that, for all x, w, we have $\sum_\chi \chi(xw)\overline{\chi(t)} = q - 1$ if $xw \equiv t \pmod q$ and 0 otherwise. Hence:

$$\begin{aligned} T(\mathcal{X}, \mathcal{W}; t) &= \frac{1}{q-1} \sum_{\chi} \sum_{(x,w) \in \mathcal{X} \times \mathcal{W}} \chi(xw)\overline{\chi(t)} \\ &= \frac{XW^*}{q-1} + \frac{1}{q-1} \sum_{\chi \neq \chi_0} \sum_{(x,w) \in \mathcal{X} \times \mathcal{W}} \chi(xw)\overline{\chi(t)}, \end{aligned}$$

by putting aside the contribution of the principal character χ_0. Write that equality as $T(\mathcal{X},\mathcal{W};t) = \frac{XW^*}{q-1}\cdot\big(1+V^*(t)\big)$. We then have:

$$V^*(t) = \frac{1}{XW^*}\sum_{\chi\neq\chi_0}\sum_{(x,w)\in\mathcal{X}\times\mathcal{W}}\chi(xw)\overline{\chi(t)},$$

and we can express the sum of the squared deviations $\big|V^*(t)\big|^2$ as:

$$\sum_{t\neq 0}\big|V^*(t)\big|^2 = \frac{1}{(XW^*)^2}\sum_{\chi,\chi'\neq\chi_0}\sum_{x,w,x',w'}\chi(xw)\overline{\chi'(x'w')}\sum_{t\neq 0}(\chi\overline{\chi'})(t).$$

The sum over t on the right-hand side is equal to $q-1$ if $\chi=\chi'$ and vanishes otherwise, so that:

$$\sum_{t\neq 0}\big|V^*(t)\big|^2 = \frac{q-1}{(XW^*)^2}\sum_{\chi\neq\chi_0}\sum_{x,w,x',w'}\chi(xw)\overline{\chi'(xw)} = \frac{q-1}{(XW^*)^2}\sum_{\chi\neq\chi_0}\big|S(\chi)\big|^2,$$

where $S(\chi)=\sum_{x\in\mathcal{X}}\chi(x)\sum_{w\in\mathcal{W}}\chi(w)$. Now since $\mathcal{X}$ is an interval of the form $[1,X]$, it is classical that GRH implies, for any $\delta>0$, $\big|\sum_{x\in\mathcal{X}}\chi(x)\big|\leq c_\delta X^{1/2}q^\delta$ for some constant $c_\delta>0$ (see e.g. [Mon71, Eq. (13.2)]). Hence:

$$\sum_{t\neq 0}\big|V^*(t)\big|^2 \leq \frac{q-1}{(XW^*)^2}\cdot c_\delta^2 X q^{2\delta}\cdot\sum_{\chi}\sum_{(w,w')\in\mathcal{W}^2}\chi(w)\overline{\chi(w')} \leq \frac{c_\delta^2 q^{2\delta}(q-1)^2}{XW^*}$$

by using orthogonality again. Then, the Cauchy–Schwarz inequlity yields:

$$\sum_{t\neq 0}\big|V^*(t)\big| \leq \sqrt{\frac{c_\delta^2 q^{2+2\delta}}{XW^*}}\cdot\sqrt{q-1} \leq \frac{c_\delta q^\delta (q-1)^{3/2}}{\sqrt{XW}}.$$

Finally, observe that for $t\neq 0$, we have:

$$\begin{aligned} V(\mathcal{X},\mathcal{W};t) &= \frac{q}{XW}T(\mathcal{X},\mathcal{W};t) - 1 = \frac{q}{XW}\cdot\frac{XW^*}{q-1}\cdot\big(1+V^*(t)\big) - 1 \\ &= \frac{qW^*}{(q-1)W}V^*(t) - \frac{W-q(W-W^*)}{(q-1)W}. \end{aligned}$$

On the last line, the first term is bounded in absolute value by $\frac{q}{q-1}\big|V^*(t)\big|$, and the second term by $\frac{q}{q-1}W$. As a result, we get:

$$\sum_{t\in\mathbb{Z}/q\mathbb{Z}}\big|V(\mathcal{X},\mathcal{W};t)\big| \leq \frac{q}{q-1}\sum_{t\neq 0}\big|V^*(t)\big| + \frac{q}{W} + \big|V(0)\big| \leq \frac{c_\delta q^{3/2+\delta}}{\sqrt{XW}} + \frac{2q}{W}$$

which yields the stated result for $\kappa_\delta = c_\delta + 2$, say (as a coarse upper bound). ■

We now discuss our key statistical lemmas. The first one ensures that the faulty signature $S' = y^d\cdot\alpha_p + \big(a + r'(y-a^e)\big)\cdot\alpha_q$ is indistinguishable from a uniform random element in $\mathbb{Z}/N\mathbb{Z}$ if the nonce r' is large enough. We write x instead of r' in the rest of this section.

Lemma 5.2. *Let $N = pq$ be a n-bit balanced RSA modulus and e the public exponent, $0 \leq y < 2^{n-1}$ a random integer and x a random nonzero ρ-bit integer. Fix an arbitrary integer a. Assuming that the Generalized Riemann Hypothesis holds, the statistical distance between the distribution of $S' = y^d \cdot \alpha_p + \big(a + x(y - a^e)\big) \cdot \alpha_q \bmod N$ and the uniform distribution modulo N is bounded as:*

$$\Delta_1(S') \leq \kappa_\delta q^\delta \sqrt{\frac{N}{XY}} \leq 2\kappa_\delta \cdot 2^{(\delta n - \rho)/2}$$

for any $\delta > 0$, with κ_δ as in Lemma 5.1.

Proof. The statistical distance between the distribution of S' and the uniform distribution can be written as:

$$\Delta_1(S') = \frac{1}{2} \sum_{(s,t) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}} \left| \Pr_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \begin{bmatrix} S' \equiv s & (\bmod\ p) \\ S' \equiv t & (\bmod\ q) \end{bmatrix} - \frac{1}{N} \right|$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the integer intervals $[1, X]$ and $[0, Y)$ with $X = 2^\rho - 1$ and $Y = 2^{n-1}$ respectively. Let us estimate the probability

$$P(s,t) = \Pr_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \begin{bmatrix} S' \equiv s & (\bmod\ p) \\ S' \equiv t & (\bmod\ q) \end{bmatrix}$$

appearing in that equation for some fixed $(s,t) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$.

We have $S' \equiv s \pmod p$ if and only if $y^d \equiv s \mod p$, i.e. $y \equiv s^e \mod p$. Hence, the solutions of the first congruence are of the form $y = (s^e \bmod p) + pw$ for w in the integer interval $[0, W_s)$, $W_s = \lceil \frac{Y - (s^e \bmod p)}{p} \rceil$. Then, the second equation, which is equivalent to $a + x(y - a^e) \equiv t \pmod q$, becomes $x(pw + (s^e \bmod p) - a^e) \equiv t - a \pmod q$. This can be written in the form $x(w + w_0) = t_0 \pmod q$, with $w_0 = \frac{(s^e \bmod p) - a^e}{p} \bmod q$ and $t_0 = \frac{t-a}{p} \bmod q$. The number of solutions (x, w) is thus $T(\mathcal{X}, \mathcal{W}_s; t_0)$, with $\mathcal{W}_s = [w_0, w_0 + W_s)$. Hence:

$$P(s,t) = \frac{1}{XY} T\big(\mathcal{X}, \mathcal{W}_s; t_0\big) = \frac{W_s}{qY} + \frac{W_s}{qY} V\big(\mathcal{X}, \mathcal{W}_s; t_0\big).$$

Note that $\mathcal{W}_s$ depends only on s (not on t), and that $t \mapsto t_0$ is a permutation of $\mathbb{Z}/q\mathbb{Z}$. Thus, for fixed s, we can sum the previous equation over $t \in \mathbb{Z}/q\mathbb{Z}$, which gives:

$$\sum_{t \in \mathbb{Z}/q\mathbb{Z}} \left| P(s,t) - \frac{1}{N} \right| \leq q \cdot \left| \frac{W_s}{qY} - \frac{1}{N} \right| + \frac{W_s}{qY} \sum_{t \in \mathbb{Z}/q\mathbb{Z}} \big| V\big(\mathcal{X}, \mathcal{W}_s; t\big) \big|.$$

Now $Y/p - 1 \leq W_s \leq Y/p + 1$, so that the first term on the right-hand side is bounded by $1/Y$. Thus, Lemma 5.1 yields:

$$\sum_{t \in \mathbb{Z}/q\mathbb{Z}} \left| P(s,t) - \frac{1}{N} \right| \leq \frac{1}{Y} + \frac{W_s}{qY} \cdot \frac{\kappa_\delta q^{3/2+\delta}}{\sqrt{XW_s}} = \frac{\kappa_\delta q^{1/2+\delta}}{\sqrt{XY \cdot p/2}}$$

using the coarse upper bound $W_s/Y \leq 2/p$. Summing further over s, we finally obtain:

$$\sum_{s,t} \left| P(s,t) - \frac{1}{N} \right| \leq \frac{p}{Y} + \kappa_\delta q^\delta \sqrt{\frac{2N}{XY}}$$

and hence the desired result, since $p \leq \sqrt{N}$ and $Y > X$.

■

Remark. Concretely, this result means that, for large enough N, it suffices to take ρ slightly larger than a given ε to obtain a statistical distance of $2^{-\varepsilon}$.

If we do not want to rely on the Riemann Hypothesis, we can obtain an unconditional bound by replacing the use of GRH in Lemma 5.1 by the Pólya–Vinogradov inequality (or the Burgess bound). However, statistical indistinguishability from uniform then requires somewhat larger values of ρ: at least $n/4+\varepsilon+o(1)$ with Pólya–Vinogradov or $n/8+\varepsilon+o(1)$ with the Burgess bound.

The security proof requires another statistical lemma which ensures that the adversary has a negligible probability of querying the correct value $\omega \leftarrow \mathcal{H}(M,r)$ given a faulty signature.

Lemma 5.3. *Let $N, e, a, \delta, \kappa_\delta$ be as in Lemma 5.2. Assume that $\rho \geq k_h + \delta n + 2\log_2(4\kappa_\delta)$. For any choice of $S' \in \mathbb{Z}/N\mathbb{Z}$, except possibly a negligible fraction $2^{-k_h/2}$ of them, and any k_h-bit value ω', the probability that a solution $(x,y) \in [1,2^\rho) \times [0,2^{n-1})$ of the equation $S' \equiv y^d \cdot \alpha_p + \big(a + x(y - a^e)\big) \cdot \alpha_q \pmod N$ satisfies that the most significant k_h bits $\omega \in [0,2^{k_h})$ of y coincides with ω' is bounded as:*

$$\Pr\left[\omega = \omega' | S'\right] \leq \frac{3}{2^{k_h}}.$$

Remark. Concretely, this result means that we must choose ρ larger than k_h.

Proof. Suppose $\mathcal{Y}$ is of the form $[0,Y)$ with $Y = mY_0$ for some integers $m, Y_0 \geq 1$, and write $\mathcal{Y}_\omega = [\omega Y_0, (\omega+1)Y_0)$ for $\omega = 0, 1, \ldots, m-1$. Let us denote by $T_\omega(S')$ the number of solutions x, y of the equation $S' \equiv y^d \cdot \alpha_p + \big(a + x(y - a^e)\big) \cdot \alpha_q \pmod N$ such that $(x,y) \in \mathcal{X} \times \mathcal{Y}_\omega$. We have:

$$\Pr\left[y \in \mathcal{Y}_\omega \mid (x,y) \in \mathcal{X} \times \mathcal{Y} \text{ and } S' \equiv y^d \cdot \alpha_p + \big(a + x(y - a^e)\big) \cdot \alpha_q \pmod N\right] = \frac{T_\omega(S')}{T(S')}.$$

Since $T(S') = XY \cdot P(s,t)$ with $P(s,t)$ as in Lemma 5.2, we have, thanks to this lemma, for all values of S' except possibly a negligible fraction $2^{-k_h/2}$ of them:

$$\left|T(S') - \frac{XY}{N}\right| \leq XY \cdot 2\kappa_\delta \cdot 2^{(\delta n - \rho)/2} \cdot 2^{k_h/2} \leq \alpha \cdot XY_0,$$

with $\alpha = 2\kappa_\delta \cdot 2^{(\delta n - \rho + k_h)/2}$. A proof similar to the one used for Lemma 5.2 leads to $\left|T_\omega(S) - XY_0/N\right| \leq \alpha \cdot XY_0$. We thus get:

$$\frac{1-\alpha}{m+\alpha} = \frac{\frac{XY_0}{N} \cdot (1-\alpha)}{\frac{XY}{N} \cdot (1+\alpha/m)} \leq \frac{T_\omega(S')}{T(S')} \leq \frac{\frac{XY_0}{N} \cdot (1+\alpha)}{\frac{XY}{N} \cdot (1-\alpha/m)} = \frac{1+\alpha}{m-\alpha}.$$

It follows that:

$$-\frac{(m+1)\alpha}{m(m+\alpha)} = \frac{1-\alpha}{m+\alpha} - \frac{1}{m} \leq \frac{T_\omega(S)}{T(S')} - \frac{1}{m} \leq \frac{1+\alpha}{m-\alpha} - \frac{1}{m} = \frac{(m+1)\alpha}{m(m-\alpha)}.$$

And finally, if we bound $(m+1)/m$ above by 2 and α by $1/2$ (i.e. $\rho \geq k_h + \delta n + 2\log_2(4\kappa_\delta)$), we obtain:

$$\left|\frac{T_\omega(S')}{T(S')} - \frac{1}{m}\right| \leq \frac{2\alpha}{m - 1/2} = \frac{4\alpha}{2m-1} \leq \frac{4\alpha}{m}.$$

Thus, $\Pr\left[\omega = \omega' | S'\right] \leq \frac{4\alpha}{m} + \frac{1}{m} \leq \frac{3}{2^{k_h}}$ with $m = 2^{k_h}$. ∎

Algorithm 5.2 Initial transition: extending state.

1: **oracle** $\mathcal{H}(m, r)$	1: **oracle** $\mathcal{H}_0(m, r)$
2: **if** $(m, r) \notin \mathsf{dom}(h)$ **then**	2: **if** $(m, r) \notin \mathsf{dom}(h)$ **then**
3: $h[m, r] \leftarrow \{0,1\}^{k_h}$	3: $\omega \leftarrow \{0,1\}^{k_h}$
4: **end if**	4: $h[m, r] \leftarrow (\omega, c, \perp)$
5: **return** $h[m, r]$	5: **end if**
	6: **return** $\pi_1(h[m, r])$

$$\Pr[\mathcal{UF}\text{-}\mathcal{CMA}^{\mathcal{A},\mathcal{K},\mathcal{S},\mathcal{F},\mathcal{V}} : win] = \Pr[\mathit{Game0} : win]$$

5.4 Security proof

The sequence of games presented in this Section and formal justifications for all transitions between games are formalized in EasyCrypt. However, Lemmas 5.2 and 5.3 are stated as axioms of the formalization. Formally proving these lemmas is outside the scope of this work, as it would first require to formalize at least those properties of additive characters used in our proof.

The hash functions $\mathcal{G}$ and $\mathcal{H}$ are modelled as random oracles. For clarity, we display the initial definition of $\mathcal{H}$ on the left in Fig. 5.2. The initial definition of $\mathcal{G}$ is similar. We assume two global maps h and g are used to build the random oracles. Our proof works mostly by transforming the random oracle $\mathcal{H}$. We therefore display the code for $\mathcal{H}$ for each transition, only displaying other oracles when they suffer non-trivial changes.

Game 0

We initially transform both random oracles to keep track of the first caller to make a particular query. It can be either the adversary (Adv), the signature oracle (Sig), or the faulty signature oracle (FSig). Calls made by the experiment when checking the validity of the forgery do not need to tag their query as they are the last queries made to the random oracles and do not need to update its state. We also extend the internal state of $\mathcal{H}$ with an additional field for use later in the proof, and currently set to a default value $\perp$.

Games 1 and 2

In Game 1, we anticipate a call to $\mathcal{G}$ on the output of $\mathcal{H}$ every time $\mathcal{H}$ is called. When $\mathcal{H}$ is called by either one of the signing oracles, we return the result of that call to $\mathcal{G}$ as well as the result of the current $\mathcal{H}$ query, allowing broad simplifications to the signing oracles. In Game 2, we deal with collisions on r and ω values in the signing oracles. In later steps of the proof, we will need the control-flow of the faulty signature oracle to be completely independent from both r and ω, and we modify the oracle to allow these later transformations. Fresh queries are treated normally. Non-fresh queries made by the signing oracles are resampled as fresh if the previous query had been made by the faulty signature oracle. Non-fresh queries made by the faulty signature oracle are resampled, but not stored into the state. Game 1 is perfectly indistinguishable from Game 0, and Game 2 can be distinguished from Game 1 if either i. (lines 2, 5 and 6) the fresh r used in $\mathcal{H}$-queries made by the signing oracles collides with a previously used r (with probability at most $(q_{\mathcal{S}} + q_{\mathcal{F}}) \cdot (q_{\mathcal{H}} + q_{\mathcal{S}} + q_{\mathcal{F}}) \cdot 2^{-k_0}$); ii. (lines 4, 7 and 8) or the fresh ω used in $\mathcal{G}$-queries made by the signing oracles collides with a previously used ω (with probability at most

Algorithm 5.3 Games 1 and 2: anticipating calls to $\mathcal{G}$ and removing signing collisions.

```
1: oracle H_1(c, m, r)
2:     if (m, r) ∉ dom(h) then
3:         ω ← {0,1}^{k_h}
4:         h[m, r] ← (ω, c, ⊥)
5:         st ← G(c, ω)
6:     else
7:         ω ← π_1(h[m, r])
8:         if c = Adv then
9:             st ← ⊥
10:        else
11:            st ← G(c, ω)
12:        end if
13:    end if
14:    return (ω, st)
```

```
1: oracle H_2(c, m, r)
2:     if (m, r) ∉ dom(h) ∨ c = FSig ∨
       (c = Sig ∧ π_2(h[m, r]) = FSig)
   then
3:         ω ← {0,1}^{k_h}
4:         st ← {0,1}^{k_g}
5:         if c ≠ FSig ∨ (m, r) ∉ dom(h)
   then
6:             h[m, r] ← (ω, c, ⊥)
7:         end if
8:         if c ≠ FSig ∨ ω ∉ dom(g) then
9:             g[ω] ← (st ⊕ (r || 0^{k_g − k_0}), c)
10:        end if
11:    else
12:        ω ← π_1(h[m, r])
13:        if c = Adv then
14:            st ← ⊥
15:        else
16:            (ω, st) ← ⊥
17:        end if
18:    end if
19:    return (ω, st)
```

$$\Pr[\mathit{Game0} : \mathit{win}] \leq \Pr[\mathit{Game2} : \mathit{win}] + (q_{\mathcal{H}} + q_{\mathcal{S}} + q_{\mathcal{F}}) \cdot \left(\frac{q_{\mathcal{S}} + q_{\mathcal{F}}}{2^{k_0}} + \frac{q_{\mathcal{G}} + q_{\mathcal{H}} + q_{\mathcal{S}} + q_{\mathcal{F}}}{2^{k_h}} \right)$$

$(q_{\mathcal{H}} + q_{\mathcal{S}} + q_{\mathcal{F}}) \cdot (q_{\mathcal{G}} + q_{\mathcal{H}} + q_{\mathcal{S}} + q_{\mathcal{F}}) \cdot 2^{-k_h})$. Note that the value stored in $g[\omega]$ at line 9 in $\mathcal{H}_2$ is uniformly distributed since st is.

Game 3

Given that $\mathcal{H}$ now samples both bitstrings that compose the final padded message, we compute the entire signature in $\mathcal{H}$ when called by either one of the signing oracles. We transform the experiment to sample an integer x^* and compute $y^* = x^{*e} \bmod N$ to serve as one-way challenge. We embed it in the state when replying to $\mathcal{H}$ queries made by the adversary. Everything up to this point has been set up so that the signing oracles can simply use $\pi_3(h[m, r])$ as the padded message for m with salt r. Game 3 includes this simplification. We introduce additional notation for clarity in the rest of the proof. Consider the function:

$$f_{(e,N),y^*,c} : S \mapsto \begin{cases} y^* \cdot S^e \bmod N & \text{if } c = \mathsf{Adv} \\ S^e \bmod N & \text{otherwise} \end{cases}$$

For a set $X \subseteq \mathbb{Z}/N\mathbb{Z}$, we denote by $\mathsf{pim}_{(e,N),y^*,c}(X)$ the uniform distribution on the set $S = \{S \in \mathbb{Z}/N\mathbb{Z} \mid f_{(e,N),y^*,c}(S) \in X\}$.

Game 3 is indistinguishable from Game 2 exactly when x^* is invertible. Therefore, the probability that the adversary distinguishes the two games is exactly $\frac{p+q-1}{p \cdot q}$. We have

Algorithm 5.4 Games 3 and 4: Embedding one-way challenge and oracle queries in $\mathcal{F}$.

```
1:  oracle H_3(c, m, r)
2:      if (m, r) ∉ dom(h) ∨ c = FSig ∨ (c = Sig ∧ π_2(h[m, r]) = FSig) then
3:          S ← pim_{(e,N),y*,c}([0..2^{n-1}])
4:          y ← f_{(e,N),y*,c}(S)
5:          b || ω || st ← i2osp(y)
6:          if c ≠ FSig ∨ (m, r) ∉ dom(h) then
7:              h[m, r] ← (ω, c, S)
8:          end if
9:          if c ≠ FSig ∨ ω ∉ dom(g) then
10:             g[ω] ← (st ⊕ (r || 0^{k_g − k_0}), c)
11:         end if
12:     else
13:         ω ← π_1(h[m, r])
14:         if c = Adv then
15:             st ← ⊥
16:         else
17:             (ω, st) ← ⊥
18:         end if
19:     end if
20:     return (ω, st)
```

```
1:  oracle H_4(c, m, r)
2:      if (m, r) ∉ dom(h) ∨ c = FSig then
3:          S ← pim_{(e,N),y*,c}([0..2^{n-1}])
4:          y ← f_{(e,N),y*,c}(S)
5:          b || ω || st ← i2osp(y)
6:          if c ≠ FSig then
7:              h[m, r] ← (ω, c, S)
8:              g[ω] ← (st ⊕ (r || 0^{k_g − k_0}), c)
9:          end if
10:     else
11:         ω ← π_1(h[m, r])
12:         if c = Adv then
13:             st ← ⊥
14:         else
15:             (ω, st) ← ⊥
16:         end if
17:     end if
18:     return (ω, st)
```

$\Pr[\mathit{Game2} : \mathit{win}] \leq \Pr[\mathit{Game3} : \mathit{win}] + 2^{-\frac{n}{2}+2}$

$\Pr[\mathit{Game3} : \mathit{win}] \leq \Pr[\mathit{Game4} : \mathit{win}] + \frac{q_{\mathcal{H}} \cdot q_S}{2^{k_0}} + \frac{q_{\mathcal{G}} \cdot q_{\mathcal{F}} \cdot 3}{2^{k_h}} + \frac{q_{\mathcal{G}} \cdot q_{\mathcal{F}}}{2^{k_h/2}}$

$p + q - 1 \leq 2^{\frac{n}{2}+1}$ and $2^{n-1} \leq p \cdot q$ and we can therefore bound the probability of this simulation failing by $2^{-\frac{n}{2}+2}$. Since the invertibility of x^* is important in some later steps, we in fact let $\mathcal{H}$ compute a response only when x^* is invertible. In the inverter, since x^* is not public, we instead check the invertibility of y^*, which is equivalent. For simplicity, we omit discussions regarding this detail in the rest of this section.

Game 4

In this game, we stop keeping track of the random oracle queries made by the faulty signature oracle. This is an important step towards being able to apply Lemma 5.2, which only discusses the statistical distance between two distributions on S', rather than (ω, S'). Note that, in Coron and Mandal's proof, Lemma 5.2 is applied before this transition, in a context in which its premises are not fulfilled. By removing data about random oracle queries, we introduce observable changes in the game's behaviour whenever the adversary queries $\mathcal{H}$ with an r that was used previously in a faulty signature query, or whenever the adversary queries $\mathcal{G}$ with an ω that was used previously in a faulty signature query. We bound the probability of the adversary guessing an ω value using Lemma 5.3. Since the view of the adversary does not depend on r values sampled by the faulty signature oracle

Algorithm 5.5 Game 5: sampling faulty signatures.

```
1: oracle F_4(m, ε, a)
2:     r ← {0,1}^{k_0}
3:     S ← pim_{(e,N),y*,c}([0..2^{n-1}))
4:     y ← S^e mod N
5:     r' ← {0,1}^ρ \0
6:     S' ← y^d * α_p + (a + (y − a^e)) * α_q
7:     return i2osp(S')
```

```
1: oracle F_5(m, ε, a)
2:     r ← {0,1}^{k_0}
3:     S' ← [0..N)
4:     return i2osp(S')
```

$$\Pr[\mathit{Game4} : \mathit{win}] \leq \Pr[\mathit{Game5} : \mathit{win}] + q_{\mathcal{F}} \cdot 2 \cdot \kappa_\delta \cdot 2^{\frac{\delta n - \rho}{2}}$$

(see Fig. 5.5), the probability of the adversary guessing an r value used in generating a faulty signature is easily bounded.

Game 5

Our main goal at this stage is to show that faulty signatures are in fact indistinguishable from uniform randomness and can be simulated without using the random oracles. Once this is done, we will be able to resume the proof of security following more standard PSS proofs.

We now use Lemma 5.2 to completely simulate faulty signature oracle queries. We focus on the faulty signature oracle, inlining and simplifying $\mathcal{H}$ knowing that $c = \mathsf{FSig}$. On the left, we display the simplified faulty signature oracle from Game 4 for reference. We make use of elementary properties of the statistical distance and Lemma 5.2 to bound the probability of distinguishing Games 5 and 6. Note that sampling S in $\mathsf{pim}_{(e,N),y^*,c}([0..2^{n-1}))$ and applying the public RSA permutation to obtain y is perfectly equivalent to sampling y in $[0..2^{n-1})$. In the bound, the δ and κ_δ are as in Lemma 5.1.

Game 6

With the faulty signature oracle simplified away, we can now focus on simulating the signature oracle. From now on, the c argument to $\mathcal{H}$ can no longer be FSig. More generally, it is impossible for any entry in h or g to be tagged with FSig. The signature oracle we have defined at this point is not a valid simulator as it does not run in polynomial time. To ensure that it does, we replace the sampling operation at line 3 in Fig. 5.4 (right) with the loop displayed on the left of Fig. 5.6 to sample S. The adversary can distinguish the two games whenever the loop finishes in a state where y does not start with a 0 bit. At each iteration of the loop, the S sampled is invalid with probability at most $\frac{1}{2}$. The probability that *all* iterations produce an invalid S is therefore bounded by $\frac{1}{2^{k_0}}$, since all samples are independent. $\mathcal{H}_7$ may now be queried $q_{\mathcal{H}} + q_{\mathcal{S}}$ times, allowing us to conclude.

Reduction

All the oracles are simulated without using any secret data. We now focus on building an inverter. The adversary can win in two disjoint cases:

- either the $\mathcal{H}$-query made by the verification algorithm is fresh (this occurs with probability at most 2^{-k_h}),

Algorithm 5.6 Game 6 and inverter: sampling S in polynomial time.

```
1: while (!0 ≤ y < 2^{n-1}) ∧ i < k_0 do
2:     S ← [0..N)
3:     y ← f_{(e,N),y*,c}(S)
4:     i ← i + 1
5: end while
```

```
1: oracle I(e, N, y*)
2:     (m, s) ← A^{H_7,G_7,S_7,F_7}(e, N)
3:     S ← os2ip(s)
4:     y ← S^e mod N
5:     b || ω || st ← i2osp(y)
6:     r || γ ← st ⊕ g[ω]
7:     (ω', Adv, u) ← h[m, r]
8:     return S · u^{-1}
```

$\Pr[\mathit{Game5} : \mathit{win}] \leq \Pr[\mathit{Game6} : \mathit{win}] + \frac{q_{\mathcal{H}}+q_S}{2^{k_0}}$

$\Pr[\mathit{Game6} : \mathit{win}] \leq \Pr[\mathcal{OW}\text{-}\mathcal{RSA}^{\mathcal{I}} : x = x^*] + \frac{1}{2^{k_h}} + \frac{q_{\mathcal{H}}}{2^{\frac{n}{2}-1}}$

- or the $\mathcal{H}$-query made by the verification algorithm was previously made by the adversary. If the query was made by the signature oracle, the forgery cannot be fresh and the adversary cannot win.

In the latter case, the one-way challenge can then be recovered by the inverter shown on the right of Fig. 5.6. The key observation is that, in case of a successful forgery, we have $y = S^e \bmod N$ (line 4) and $y = y^* \cdot u^e \bmod N$ (by invariant on h). By definition of y^* and the morphism and injectivity properties of RSA, we therefore have $S = x^* \cdot u$. We need to also consider the case where a value u stored in the h map by the adversary is not invertible, which occurs with probability at most $q_{\mathcal{H}} \cdot 2^{-n/2+1}$.

The final bound is obtained by transitively using the individual transition bounds.

PART II

SECURITY OF IMPLEMENTATIONS OF THE ECDSA SCHEME, EFFICIENCY OF THE SCHEME ELLIGATOR SQUARED

The use of elliptic curves in cryptography has first been proposed independently by Koblitz [Kob87] and Miller [Mil85] in 1985. Elliptic curves offer many advantages for public-key cryptography compared to more traditional settings like RSA and finite field discrete logarithms, including higher efficiency, a much smaller key size that scales gracefully with security requirements, and a rich geometric structure that enables the construction of additional primitives like bilinear pairings. On the Internet, adoption of elliptic curve cryptography is growing in general-purpose protocols like TLS, SSH and S/MIME, as well as anonymity and privacy-enhancing tools like Tor (which favors ECDH key exchange in recent versions) and Bitcoin (which is based on the ECDSA signature scheme).

In this second part we present some contributions to two areas of elliptic curve cryptography. First we analyze in Chapter 6 the security of implementations of the ECDSA signature scheme, including fast ones using GLV/GLS decomposition. In some cases we prove the security of the scheme whereas in other cases, we present different attacks, using fault injection or a biased value or even a side-channel analysis. Second, we efficiently implement in Chapter 7 the scheme Elligator Squared the purpose of which is to represent uniformly random points on elliptic curves as uniformly random bit strings. This implementation is performed on a fast binary curve and we demonstrate that the overhead due to this uniform bit string representation, which enhances anonymity and privacy, is quite minimal.

Contents

Chapter 6
GLV/GLS Decomposition and Security of Implementations of the ECDSA Scheme

6.1 Introduction

The fastest implementations of elliptic curve cryptography in recent years have been achieved on curves endowed with nontrivial efficient endomorphisms, using generalizations of the techniques due to Gallant–Lambert–Vanstone (GLV) and Galbraith–Lin–Scott (GLS). In such implementations, a scalar multiplication of a curve point P of prime order n is computed as a double multiplication $[k_1]P + [k_2]\psi(P)$ for k_1, k_2 half-size scalars and ψ an efficient curve endomorphism: this evaluates to $(k_1 + k_2\lambda)P$ for some constant λ.

In a protocol that needs to generate a random multiple of P, a first step common to all GLV/GLS-like implementations is to generate the half-size scalars k_1 and k_2. Several approaches have been suggested to do so, which can be roughly divided in two families: those in which k_1 and k_2 are themselves generated at random, and those in which a full-size scalar k is first selected at random and then decomposed as $k = k_1 + k_2\lambda$. The main goal of this chapter is to discuss security issues that may arise during this decomposition step using either approach.

On the one hand, if k_1 and k_2 are chosen uniformly at random in $[0, \sqrt{n})$ (using rejection sampling, say), we show that, in the quadratic GLS setting, the resulting scalar $k = k_1 + k_2\lambda$ is statistically close to uniform, and hence resulting protocols are secure. However, if they are chosen uniformly at random integers of $\lfloor \frac{1}{2} \log_2 n \rfloor$ bits, the resulting k is slightly skewed, and hence not suitable for use as a nonce in Schnorr-like signature schemes like ECDSA, for example.

Indeed, for GLS curves, we show that this results in a bias of up to 1 bit on a suitable multiple of $k \bmod n$, and that this bias is practically exploitable: while lattice-based attacks cannot exploit a single bit of bias, we demonstrate that an earlier attack strategy by Bleichenbacher makes it possible. In doing so, we set a record by carrying out the *first ECDSA full key recovery using a single bit of bias* (on both a GLS curve and a standard prime field SEC curve).

On the other hand, computing k_1 and k_2 by decomposing a uniformly random $k \in [0, n)$ solves all problems of statistical bias, but the decomposition algorithm itself is liable to leak side-channel information. Early proposed algorithms to do so relied on lattice reduction and exhibited a significant amount of timing channel leakage. More recently, constant-time approaches have also been proposed, but we show that they are amenable to power analysis: we describe a template attack that can be combined with classical lattice-based attacks on ECDSA to achieve full key recovery on physiscal devices.

Finally, we present some fault attacks for implementations of ECDSA based on a par-

ticular implementation of the scalar multiplication. Theses attacks have been found thanks to our tool described in Chapter 4.

This work, except the part concerning the fault attacks, is to be presented at Asiacrypt 2014 [AFG+14].

6.1.1 The GLV/GLS techniques

Many record implementations of elliptic curve cryptography in software, including, most recently, works such as [OLARH14, BCHL13, CHS14], rely on elliptic curves endowed with fast endomorphisms, as constructed by the methods due to Gallant–Lambert–Vanstone (GLV) [GLV01], Galbraith–Lin–Scott (GLS) [GLS11], and generalizations thereof. In such implementations, the fast endomorphism ψ on the elliptic curve $E/\mathbb{F}_q$ is used to speed up full size scalar multiplications $[k]P$ by computing them as multi-exponentiation $[k_1]P + [k_2]\psi(P)$, where k_1 and k_2 are roughly half of the size of k. Indeed, on a prime order subgroup of $E(\mathbb{F}_q)$, ψ acts by multiplication by some constant λ, and thus, for a generator P of that subgroup, we have $[k_1]P + [k_2]\psi(P) = [k_1 + k_2\lambda]P$.

In order to compute random scalar multiplications with those techniques, two types of approaches have been considered, as far back as in the earliest presentations of the GLV method (such as Gallant's talk at ECC'99 [Gal99]).

On the one hand, k_1 and k_2 can simply be chosen uniformly at random in a suitable half-length interval. This appraoch, which we call the *recomposition technique* (since k is "recomposed" as $k = k_1 + k_2\lambda$), results in a very simple implementation, and has been used in several implementation records including [OLARH14], but Gallant expressed concerns about possible biases in the resulting scalar k. Such concerns have been partially vindicated by some numerical evidence provided by Brumley and Nyberg [BN09], who also described a relatively general way to choose intervals for k_1 and k_2 so that the resulting choice of k is in fact secure (in the sense that it has high entropy). However, the Brumley–Nyberg method is a bit cumbersome, and no attack so far has been demonstrated against arbitrary half-length uniform choices of k_1 and k_2, so that the security picture is somewhat unclear.

On the other hand, one can also pick k at random and subsequently deduce half-length values k_1 and k_2, which eliminates concerns regarding possible biases in the distribution of k. This *decomposition technique* usually relies on lattice reduction in dimension 2 (or equivalently, continued fractions, a generalized Euclidean algorithm, etc.), as originally described in the GLV paper [GLV01], and is significantly more computationally demanding than recomposition. Simplifications of this method have later been proposed (particularly in [PJKL02]), as well as higher-dimensional generalizations [NS04] to tackle decompositions involving several endomorphisms (as recently used in [Smi13, GI13] for instance).

6.1.2 ECDSA attacks

Bias exploitation

The success of GLV/GLS method in implementations lately makes it desirable to reconsider these decomposition and recomposition techniques from a security viewpoint. We do so in this chapter in the context of ECDSA signatures, one of the most widely deployed elliptic curve cryptographic schemes, and an interesting target for the cryptanalyst (like other Schnorr-like signature schemes) due to its sensitivity to biases in the distribution of nonce values, as demonstrated by the powerful attack due to Howgrave-Graham and Smart [HGS01] based on lattice reduction techniques, which breaks (EC)DSA when some of the most significant bits of the nonces are known. This attack was analyzed in further

details by Nguyen and Shparlinski [NS02, NS03] and carried out in practice in many contexts, including against physical devices (see e.g. [NNTW05, BH09] for some examples). The basic idea is to express the key recovery problem as an instance of the Hidden Number problem (HNP), which reduces to the closest vector problem (CVP) in a suitable lattice. Since CVP is tractable in low-dimensional lattices, many practical instances of ECDSA can be broken depending on key size and the number of leaked nonce bits. The largest problem instance broken so far is the case of 2-bit nonce leaks on 160-bit curves, tackled by Liu and Nguyen [LN13] using the most advanced known techniques for lattice reduction (BKZ 2.0 [CN11]). Breaking 2-bit leaks on 256-bit curves, or 4-bit leaks on 384-bit curves seems currently out of reach (see the discussions in [CN11, MHMP14]).

In any case, there is a hard limit to what can be achieved using lattice reduction: due to the underlying structure of the HNP lattice, it is impossible to attack (EC)DSA using a single-bit nonce leak with lattice reduction. In that case, the "hidden lattice point" corresponding to the HNP solution will not be the closest vector even under the Gaussian heuristic (see [NT12]), so that lattice techniques cannot work. To break this "lattice barrier", the only known alternate attack is an algorithm due to Bleichenbacher [Ble00] which predates the attack of Howgrave-Graham and Smart, but was generally considered of mostly theoretical interest until it was recently revisited by De Mulder et al. [MHMP14] to attack 384-bit curves. Bleichenbacher devised his attack to demonstrate a vulnerability in DSS at the time, in which DSA nonces were generated by picking a random value of ℓ_n bits, where ℓ_n is the bit length of the group order n, and then to reduce it modulo n. Bleichenbacher showed that the resulting bias could be exploited in a very interesting way, obtaining a key recovery using about 2^{41} signatures and about 2^{47} time and 2^{41} memory complexities. At that time, it was not possible to mount this attack and only simulations on reduced numbers were possible and the paper was never published.

In the first stage, Bleichenbacher's algorithm reduces the signatures from 160 bits to say 40 bits using linear combinations of the original signatures and then, during a second phase, a Discrete Fourier Transform is used to recover the most significant bits of the secret key. The bias of the reduced signatures is higher than the bias of the original signatures, that's the reason why Fourier technique is needed to extract this information. This algorithm is very similar to Blum, Kalai and Wasserman algorithms [BKW03, LF06] for solving LPN and LWE problems. For 384-bit order, the first stage of Bleichenbacher original attack is not sufficiently efficient to reduce the signatures and more advanced techniques based on LLL and BKZ are needed if the number of leaked bits is high enough [MHMP14]. The modification of the first stage is not possible if less than one bit of nonces is available and we turn back to Bleichenbacher's original attack which requires a high number of signatures.

Fault injection

Biehl, Meyer and Müller [BMM00] (see also [CJ05a] for a generalization) were among the first to consider fault attacks against elliptic curve cryptosystems; more specifically, they consider an elliptic curve variant of ElGamal encryption. Their attacks exploit some of the ideas from Boneh et al. for RSA–CRT and are cast in the setting of a high-level algorithmic description of scalar multiplication between a field element and a point in the curve. Later, Naccache, Nguyen, Tunstall and Whelan [NNTW05] exhibit fault attacks on implementations of DSA and its elliptic curve variant ECDSA. Their attack introduces a fault during the generation of the nonce k and is cast in an algorithmic setting. In contrast, more recent works [BBPS11, BJPW13, MMNT13] study fault attacks against implementations of ECDSA, based on detailed accounts of integer multiplication, scalar multiplication, and point doubling. For example, the attack on integer multiplication [BBPS11] by Barenghiet

al. works by injecting faults during the integer multiplication of a known random value and the secret key. Then, by considering the textbook multiplication implementation, they show that it is possible to recover the secret key. Finally, the attack of [MMNT13] shows that it is possible to inject a fault during the conversion from projective to affine coordinates. These two attacks show that it is beneficial to consider all steps of an implementation-level description when looking for fault attacks.

6.1.3 Our contributions

Our first contribution is the first implementation of Bleichenbacher's attack against ECDSA with a single-bit on nonce bias. We carry out this attack on the standardized SECG P160 R1 elliptic curve. On this 160-bit curve, we use 2^{33} ECDSA signatures, and achieve a full key recovery in a few hours of wall-clock time on a 64-core workstation. The most time-consuming part of the attack is the first phase, in which a sorting algorithm is executed several times. This is the first key recovery from a single bit of bias, which paves the way to new applications. We stress again that this record cannot be achieved using lattice reduction techniques based on HNP problem, since even if the HNP lattice satisfies the Gaussian heuristic, a condition for finding the hidden lattice point is that the number of known bits of the nonce must be greater than $\log_2(\sqrt{\pi e/2}) \approx 1.0471$ (hence at least 2) [NT12], irrespective of the underlying lattice reduction algorithm.

As a second contribution, we show a security proof for the recomposition method on curves obtained by the quadratic GLS method once the values k_1 and k_2 are uniformly distributed in the interval $[0, \sqrt{n})$, where n is the prime group order. We prove that the statistical distance between this distribution and the uniform distribution in $[0, n)$ is negligible. Furthermore, if k_1 and k_2 are taken at random in a small interval of the form $[0, 2^m)$, where $m = \lfloor \frac{1}{2} \log_2 n \rfloor$, the bias on the distribution on k used in Bleichenbacher's attack is negligible. However, we show that the bias of the distribution on tk where t is the trace is sufficiently large and a Bleichenbacher's attack allows to recover the secret key. We also implement this attack and the complexities are similar to the previous part.

We also study the decomposition technique proposed in GLV with the implementation described by Park et al. in [PJKL02]. To this end, we propose a very efficient side-channel attack that uses the leakage on the multiplication in order to recover some of the least significant bits of the nonces. Consequently, we can thus use lattice techniques to recover the secret key.

Finally, we discover, using our tool described in Chapter 4, several new and efficient fault attacks for implementations of ECDSA based on the implementation of scalar multiplication given in Algorithm 6.1. A first attack is based on skipping the last iterations in the computation of scalar multiplication. A second attack is based on forcing the evaluation of a conditional inside the loop executed for the computation. The largest group of attacks (containing more than 100 faulted programs) is based on faulting the implementation of the point addition operation. Each faulted signature allows us to recover the least or most significant bits of the nonce; we then finish the attack using classic techniques, and obtain the secret key from a small number of faulty signatures. We also recover an existing attack [NNTW05] that lets the faulted algorithm produce valid signatures that may nevertheless be exploited in a similar fashion.

Algorithm 6.1 Scalar Multiplication of an elliptic curve point by a field element. $[2]\cdot$ denotes point doubling operation, and $+$ denotes addition.

```
1: function ECScalMul(k,P)
2:     R_0 ← ∞
3:     for i = t down to 0 do
4:         R_0 ← [2] · R_0
5:         if k_i = 1 then R_0 ← R_0 + P
6:         end if
7:     end for
8:     return R_0
9: end function
```

6.2 Preliminaries

6.2.1 Bias definition and properties

The measurement of the bias of random variables represents a significant part of our analyses. We thus recall the definition of the bias which was proposed by Bleichenbacher in [Ble00].

Definition 6.1. *Let X be a random variable over $\mathbb{Z}/n\mathbb{Z}$. The bias $B_n(X)$ is defined as*

$$B_n(X) = E(e^{2\pi iX/n}) = B_n(X \bmod n),$$

where $E(X)$ represents the mean.
Similarly, the sampled bias of a set of points $V = (v_1, \cdots, v_L)$ in $\mathbb{Z}/n\mathbb{Z}$ is defined by

$$B_n(V) = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i v_j/n}.$$

The bias as defined above presents some useful properties we recall in Lemma 6.1.

Lemma 6.1. *Let $0 < T \leq n$ be a bound and X, Y random variables uniformly distributed on the interval $[0, T-1]$.*

(a) If X is uniformly distributed on the interval $[0, n-1]$, then $B_n(X) = 0$.

(b) If X and Y are independent, then $B_n(X+Y) = B_n(X)B_n(Y)$.

(c) $B_n(-X) = \overline{B_n(X)}$ where $\overline{a}$ denotes the conjugate of a.

(d) $B_n(X) = \frac{1}{T}\left|\frac{\sin(\pi T/n)}{\sin(\pi/n)}\right|$ and $B_n(X)$ is real-valued with $0 \leq B_n(X) \leq 1$.

(e) Let a be an integer with $|a|T \leq n$ and $Y = aX$, then $B_n(Y) = \frac{1}{T}\frac{\sin(\pi aT/n)}{\sin(\pi a/n)}$

6.2.2 ECDSA signature generation

ECDSA is a NIST standard and we describe the signature generation in Algorithm 6.2.

Algorithm 6.2 ECDSA signature. P is a base point of order n and $H : \{0,1\}^* \to [0, n-1]$ is a cryptographic hash function. The private key is an element $x \in \mathbb{Z}/n\mathbb{Z}$ and the public key is denoted by (p, n, H, P, Q) with $Q = [x]P$.

1: **function** $\text{SIGN}_{\text{ECDSA}}(m)$
2: $k \xleftarrow{\$} [0, n-1]$
3: $(u, v) \leftarrow [k]P$
4: $r \leftarrow u \bmod n$; **if** $r = 0$ **then goto** step 2;
5: $s \leftarrow k^{-1}(H(m) + rx) \bmod n$; **if** $s = 0$ **then goto** step 2;
6: **return** (r, s)
7: **end function**

6.3 Bleichenbacher's Attack on single bit bias

In this part, we present our results on an ECDSA signature generation scheme where the nonce k is 1-bit biased. We demonstrate that an attack proposed some years ago by Bleichenbacher can succeed in retrieving the secret key in about 2^{37} time and 2^{33} memory complexities given 2^{33} signatures, for 160-bit order. This attack was initially focusing on the DSA signature generation scheme but can be applied without any modification to ECDSA we consider in this chapter.

The main idea consists in using the fact that the nonces k_j are chosen from a biased random variable $\mathbf{K}$, *i.e.* k are not randomly and uniformly generated on $[0, n-1]$. Because the values k_j are biased and linked with the secret key x by the equations which are used for the signature computations, these signatures, correctly manipulated, also present a bias which will only be significant for the correct value of x. In other words the bias plays the role of the distinguisher in this attack.

Obviously, for cryptographic sizes, evaluating the bias for all values in $[0, n-1]$ is impractical. However, Bleichenbacher observed that it is possible to "broaden the peak" of the bias in such a way that, with a value close the correct value of x, the bias will remain significant. Thus the bias computations can be performed on a more sparse set of candidates thanks to the Fast Fourier Transform. In return, it requires a non-negligible work on the signatures which reduces the bias, and the attack returns an approximation of the secret key, *i.e.* its most significant bits. The attack can be iterated to retrieve more bits of the secrets and as soon as sufficiently many bits of x are known, Pollard's lambda method [Pol00] can be used to derive the remaining bits. Algorithm 6.3 presents the main steps of the attack.

6.3.1 Attack analysis

We first explain why the bias can serve as a distinguisher and while doing so explain the goal of the preprocessing phase, as it was done in [MHMP14], for the sake of completeness. For that purpose, consider S ECDSA signatures (r_j, s_j) with biased nonces k_j. We have the following relation due to step 5 of Algorithm 6.2:

$$k_j = H(m_j)s_j^{-1} + r_j s_j^{-1} x \bmod n \quad \text{for} \quad 0 \leq j \leq S-1.$$

Now let $h_j = H(m_j)s_j^{-1} \bmod n$ and $c_j = r_j s_j^{-1} \bmod n$. Then the set $\{h_j + c_j x\}_{j=0}^{S-1} = \{k_j\}_{j=0}^{S-1}$ will show a significant nonzero sampled bias. Moreover, for any $w \neq x$, the sampled bias from $V_w = \{h_j + c_j w\}_{j=0}^{S-1}$ will be relatively small. Since h_j and c_j are

Algorithm 6.3 Bleichenbacher's attack given S ECDSA signatures. The parameters S, ℓ and ι have to be chosen accordingly to the bias.

Require: S biased ECDSA signatures (r_j, s_j) computed using a single secret key x.
Ensure: The ℓ most significant bits of x.

Preprocessing
for $j = 0$ **to** $S - 1$ **do**
    $h_j \leftarrow H(m_j) \cdot s^{-1} \bmod n$
    $c_j \leftarrow r_j \cdot s_j^{-1} \bmod n$
end for
Reduction of the c_j values (Sort-and-Difference Algorithm)
$A \leftarrow [(c_j, h_j)]_{0 \leq j \leq S-1}$
for $i = 1$ **to** ι **do**
    Sort A by the c_j values ▷ $c_j \leq c_{j+1}$
    for $j = 0$ **to** $S - \iota$ **do**
        $A[j] \leftarrow A[j+1] - A[j]$ ▷ $A[j] = (c_{j+1} - c_j, h_{j+1} - h_j)$
    end for
end for
Only keep the pairs (c_j, h_j) such that $c_j < 2^\ell$
Denote by L the number of such pairs
Bias computation using the inverse FFT
$Z \leftarrow (0, \cdots, 0)$ a vector of size 2^ℓ
for $j = 0$ **to** $L - 1$ **do**
    $Z_{c_j} \leftarrow Z_{c_j} + e^{2\pi i h_j / n}$
end for
$W \leftarrow \text{iFFT}(Z)$ ▷ Inverse FFT computation. The output is also a vector of complex numbers.
Find the value m such that $|Z_m|$ is maximal
return $\text{MSB}_\ell(mn/2^\ell)$

publicly computable, we thus have a way to determine the correct value of x by testing all the value $w \in [0, n-1]$.

To have a practical test, we have to broaden the peak of the bias such that values of w close to the correct value x will also show a significant bias. The peak will be broad if the c_j are relatively small. More precisely, by denoting 2^ℓ a bound such that $0 \leq c_j < 2^\ell$, then we can find an approximation of x by evaluating the sampled bias of 2^ℓ evenly-spaced values of w between 0 and $n - 1$.

The reduction of the c_j, second phase in Algorithm 6.3, can be done using a sort-and-difference algorithm. From S pairs (c_j, h_j), we first sort them according to their first element. Then we subtract each c_j from the next largest one and we take the differences of the corresponding h_j as well. We thus obtain a list of $S - 1$ pairs (c'_j, h'_j) whose values c'_j are on average $\log(S)$ bits smaller. More details about the analysis of this reduction are given later. This reduction algorithm can be repeated in order to achieve the bound 2^ℓ.

Now let $w_m = mn/2^\ell$, with $m \in [0, 2^\ell - 1]$, be 2^ℓ evenly-spaced values between 0 and $n - 1$. For sake of clarity, we keep the notation (c_j, h_j) for the reduced pairs with $c_j < 2^\ell$

and we consider having L such pairs. Then

$$B_n(V_{w_m}) = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i(h_j + c_j mn/2^\ell)/n} = \sum_{t=0}^{2^\ell-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/n}\right)e^{2\pi i t m/2^\ell}$$
$$= \sum_{t=0}^{2^\ell-1} Z_t e^{2\pi i t m/2^\ell}$$

with $Z_t = \frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/n}$. $B_n(w_m)$ can be viewed as the inverse Fast Fourier Transform of the vector $Z = (Z_0, \cdots, Z_{2^\ell-1})$. Thus the multiple bias computations can be performed very efficiently using the FFT. From Step 17 to 20 in Algorithm 6.3, we compute this vector Z. Step 21 outputs a vector of the sampled bias for the 2^ℓ candidates, i.e. $\text{iFFT}(Z) = (B_n(V_{w_0}), B_n(V_{w_1}), \cdots, B_n(V_{w_{2^\ell-1}}))$. Finally, the value of $w_m = mn/2^\ell$ with the largest sampled bias should share its ℓ most significant bits with the secret key x.

Choosing the parameters.

We first give some properties which will help to define the parameters for the attack. We can estimate the sampled bias for a wrong candidate w_m, i.e. a value w_m which do not share some most significant bits with the secret key x. More precisely, it can be shown that for w_m either significantly larger or smaller than x, we have $B_n(V_{w_m}) \approx \frac{1}{\sqrt{L}}$, which corresponds to the average distance from the origin for a random walk on the complex plane.

The second property concerns the c_j reduction phase and gives a relation between the number of signatures S and the number of reduced pairs L.

Proposition 6.1. *Consider S ECDSA signatures of the form (c_j, h_j) and $\gamma \in \mathbb{Z}$. The percentage of signatures (c'_j, h'_j) after the first application of the sort-and-difference algorithm such that $c'_j < 2^{\log q - \log S + \gamma}$ can be approximated by $1 - e^{-2^\gamma}$.*

Lemma 6.2. *Let $X_1, \ldots, X_N$ be N independent uniformly distributed random variables over $[0,1]$, and for all i, denote by $X_{(i)}$ the i-th order statistic of the X_j's (namely, $X_{(i)}$ is the i-th smallest among the X_j's). Then, the random variables $Y_i = X_{(i+1)} - X_{(i)}$ for $i = 1, \ldots, N-1$ are identically distributed, and all follow the beta distribution $B(1, N)$, of probability density function (hereafter pdf) $f(t) = N \cdot (1-t)^{N-1}$. As a result, for any constant $\alpha > 0$, we have $\Pr[Y_i \leq \alpha/N] = 1 - e^{-\alpha} + O(1/N)$.*

Proof. Indeed, a standard formula [DN03, 2.2.1] expresses the joint pdf of $X_{(i)}$ and $X_{(i+1)}$ as:

$$f_{i,i+1}(u,v) = \begin{cases} \frac{N!}{(i-1)!(N-i-1)!} u^{i-1}(1-v)^{N-i-1} & \text{for } 0 \leq u \leq v \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the pdf f_i of Y_i is given by:

$$f_i(t) = \int_0^{1-t} f_{i,i+1}(u, u+t)du \quad \text{for } t \in [0,1].$$

The change of variable $u = (1-t)w$ gives:

$$\begin{aligned} f_i(t) &= (1-t)\int_0^1 f_{i,i+1}\big((1-t)w, (1-t)w+t\big)dw \\ &= c(1-t)\int_0^1 (1-t)^{i-1}w^{i-1}(1-w-t+wt)^{N-i-1}dw \\ &= c(1-t)^i\int_0^1 w^{i-1}(1-t)^{N-i-1}(1-w)^{N-i-1}dw \\ &= c(1-t)^{N-1}\int_0^1 w^{i-1}(1-w)^{N-i-1}dw, \end{aligned}$$

where $c = \frac{N!}{(i-1)!(N-i-1)!}$. In particular, we have $f_i(t) = c'(1-t)^{N-1}$ for some constant c' and all $t \in [0,1]$, and since $\int_0^1 f_i = 1$, we must have $f_i(t) = N(1-t)^{N-1} = f(t)$ as required. As a result, we obtain:

$$\begin{aligned} \Pr\left[Y_i \leq \frac{\alpha}{N}\right] &= \int_0^{\alpha/N} N(1-t)^{N-1}dt = 1 - \left(1-\frac{\alpha}{N}\right)^N \\ &= 1 - \exp\left(N\cdot(-\alpha/N + O(1/N^2))\right) = 1 - e^{-\alpha} + O(1/N) \end{aligned}$$

This concludes the proof. ■

As an example consider a modulus n of size 160. Starting from 2^{40} ECDSA signatures, after one iteration of the sort-and-difference algorithm, about 86.5% of them will have a value $c'_j < 2^{121}$. The percentage drops to 22.1% if we consider only those ones with a value $c'_j < 2^{118}$. Note that this proposition is only true for the first iteration of the algorithm where we really can consider variables as uniformly random and independently distributed. Clearly they are not after this: if after the first round variables were uniformly distributed, the ratio between $\gamma = -2$ and $\gamma = 1$ would be $0.125 = 1/2^3$ where it is ≈ 0.255. Sadly, it appears that the ratio progress in our disfavor when we want to iterate, *i.e.* the ratio after ι iterations is less than $(1-e^{-2^\gamma})^\iota$. We thus do not have a lower bound. However the ratio can be experimentally determined and Table 6.1 gives an overview for different values of γ up to 6 iterations.

Given S signatures, we have to choose a pair (γ, ι) such that $\log n - \iota\cdot(\log S + \gamma) = \ell$ is sufficiently small to perform a FFT in $2^\ell \log \ell$ time and 2^ℓ memory complexities. The algorithm complexity is $O\big(S\log(S) + \ell\log(\ell)\big)$. Now a verification is necessary to be sure that this set of parameters will give a successful attack. Indeed denote by $B_n(\mathbf{K})$ the initial bias which is fully determined by the number of most (or least) significant bits of the k_i which are known or set to zero (see Table 6.2 for some values). From properties (b) and (c) of the Lemma 6.1, each iteration of the sort-and-difference algorithm reduces the bias by raising it to the square of its norm (assuming that the variables are independant): indeed, let X, Y be uniformly distributed and independent random variables on $[0, n-1]$, then $B_n(X) = B_n(Y)$ and $B_n(X-Y) = B_n(X)\overline{B_n(Y)} = |B_n(X)|^2$. The final bias is then approximated by $|B_n(\mathbf{K})|^{2^\iota}$. Thus the following inequality holds since $B_n(V_{w_m}) \approx 1/\sqrt{L}$:

$$|B_n(\mathbf{K})|^{2^\iota} \gg 1/\sqrt{L},$$

where L represents as before the number of reduced pairs (c_j, h_j) with $c_j < 2^\ell$. Using Table 6.1 which gives the ratio L/S for different choices of pairs (γ, ι), we obtain a relation between S, ι, ℓ and n.

Table 6.1: Experimental ratio between the ECDSA signatures of the form (c'_j, h'_j) such that $c'_j < 2^{\log n - \iota \cdot (\log S + \gamma)}$, and the S initial signatures, after ι iterations of the sort-and-difference algorithm.

γ	-2	-1	0	1	2
1^{st} iteration	0.22	0.39	0.63	0.86	0.98
2^{nd} iteration	0.031	0.12	0.36	0.75	0.94
3^{rd} iteration	$3.2\ 10^{-3}$	0.025	0.17	0.64	0.89
4^{th} iteration	$3.0\ 10^{-4}$	$4.6\ 10^{-3}$	0.069	0.53	0.84
5^{th} iteration	$2.0\ 10^{-5}$	$6.7\ 10^{-4}$	0.022	0.40	0.79
6^{th} iteration	$2.8\ 10^{-6}$	$9.5\ 10^{-5}$	$6.5\ 10^{-3}$	0.28	0.73

Table 6.2: Some values of bias for large n, when b most (or least) significant bits of k are known, using Property (d) of Lemma 6.1.

b	1	2	3	4	5
$B_n(\mathbf{K})$	0.6366198	0.9003163	0.9744954	0.9935869	0.9983944

Note that contrary to previous reports in the literature [MHMP14, Ble00], we do not need to center the k_j around 0. Indeed sort-and-difference algorithm performs only subtractions and does not mix subtractions and additions as is common with lattice reduction or generalized birthday algorithms.

6.3.2 Implementation

We successfully implemented the attack. As our target, we chose the SECG P160 R1 curve, published in 2000 by the SECG consortium [Res00] and still considered secure. We fixed the most significant bit in the nonces and checked (with the help of the secret) that we indeed got the expected bias: ≈ 0.63662. Our C++ implementation was based on the RELIC toolkit (using its provided plain C integer arithmetic) [AG] and FFTW [FJ05]. We parallelized it in a straightforward manner (including (quick)sorting phases) and tested it on a multicore machine.

We generated 2^{33} signatures and performed 4 sort-and-difference reduction phases. 450 millions (which is 52.5%) of our initial 2^{33} signatures had their c_j reduced down to 32 bits, as was expected from table 6.1. The bias after 4 reduction steps was 0.000743558 which is slightly greater than the expected $0.63662^{2^4} \approx 0.00072792$. We then computed a FFT on 32 bits (we selected the reduced c_j smaller than 2^{32}). The best candidate had a score approximately 35% greater than the second. Both corresponding most significant bits of the secret differed only by the 31^{st} and 32^{nd} most significant bits. The 3^{rd} and 4^{th} candidates were also very close to the two first ones, with score approximately 1/3 of the best candidate. Then, there was a number of random values with maximal score

approximately 1/6 of the best one. We repeated the experiment several (5) times and got similar results, always finding at least the 30 most significant bits of the secret with the best candidate. We couldn't repeat it more because of the high computational resources involved.

The total memory used by the signatures and FFT tables was slightly more than 1 terabyte. To recover 32 bits of the private key, the attack took approximately 1150 CPU-hours, most of it being data exchange, which we can decompose as follows:

- 70%: parallelised quicksort (the most memory-intensive phase)
- 18%: signature generation (approximately 250 to 430 kilocycles per signature depending on the CPU, excluding hash computations)
- 10%: candidate selection and FFT table preparation
- $\approx$ 1%: the FFT itself.

We did not use more parallelizable sorts like Batcher odd-even mergesort [Bat68] but this would clearly be the next thing to do from a performance perspective.

Next steps of the attack to recover the following bits of the secret were done as in [MHMP14]. Basically, it amounts to a replay of algorithm 6.3 on the initial signatures, putting the previously found most significant bits of the secret into h_j. Write the private key x as $x_0 2^m + x_1$ where x_0 is the recovered m most significant bits at the first round. Then $(h_j) + (c_j)x = (h_j + c_j x_0 2^m) + (c_j x_1)$ and we want to recover the most significant bits of x_1. We proceed as in the first round, except that we now keep the c_j that are smaller than $2^{\ell+m}$ instead of 2^ℓ (thus when $\ell = m$ we just have to stop the reduction one iteration earlier). Then we build the FFT table as $Z[c_j/2^m] = Z[c_j/2^m] + e^{2\pi i h'_j/n}$. The FFT recovers the next most significant chunk of bits of the secret key. The computation restart makes it necessary to go back from the initial signatures, but there's no need to keep them in memory during the reduction. In practice we had barely enough memory to keep them, but in order to reduce memory usage they should either be stored on disk and retrieved to iterate the secret recovery, or tracked down through the reduction and rebuilt afterwards.

In practice, it is advisable to take a small security margin and reinject only 30 bits of the computed most significant bits of the secret to account for small variations of sampling around the peak. In any case, if we recurse with a wrong secret, the FFT will not detect any peak. Experiments indeed showed no peak in this case, with the highest score not being statistically different from the other ones. This paves the way for a time/memory tradeoff: suppose the hardware is limited in memory and can only work on (say) 2^{31} signatures and 2^{30} FFT size instead of the 2^{33} needed for attacking 160 bits with 4 iterations with the previous algorithm. We first reduce the c_j from 160 to 40 bits with 4 reductions as usual. We then simply guess the 10 most significant bits of the secret and build 2^{30}-sized FFT tables accordingly. The guess will be correct on the only one FFT among the 2^{10} which shows a significant peak. Since FFTs are particularly efficient, much more than sorting, this is of practical importance. Alternatively, if it's possible to compute 2^{41} signatures, we can select only the expected $1/2^{10}$ fraction of signatures whose corresponding c_j have their 10 most significant bits already zeroes, that is to say that have 150 bits instead of 160 and can be reduced to 30 with 4 iterations. Finally, since the FFT table takes less memory than the signatures (a complex number occupies 16 bytes whereas a signature requires at least 40), we could improve the attack further by either carrying out several FFTs in parallel when guessing some bits of the secret, or by increasing the size of the FFT table

slightly (with a corresponding increase of the selection bound on c_j). This would have two advantadges. Firstly, it would improve the sampling around the peak and reduce the uncertainty. Secondly, the bound increase implies that some signatures would be selected after the third round of reduction instead of the fourth, thus having a much better bias and hopefully revealing more precise information about the secret.

Our experiments targeted a 160-bit curve, but it should be pointed out that larger curves are susceptible to this attack as well. Roughly speaking, one can carry out the key recovery attack with 1-bit nonce bias on an N-bit curve in time $\approx 2^{N/5 \log_2(N/5)}$ and memory $\approx 2^{N/5}$. For example, a 256-bit curve can be attacked in time $\approx 2^{58}$ and memory $\approx 2^{52}$: generate 2^{52} signatures, perform 4 reduction steps (removing $4 \cdot 51 = 204$ bits on approximately 86% of the data), keep signatures with c_j less than 2^{52} and carry out the FFT on a table of size 2^{52}. One signature is $64 = 2^6$ bytes, so that the total memory needed for the attack is 2^{18} terabytes of storage, which corresponds to 65536 of today's 4 TB disks. This does not appear to be out of reach of well-funded adversaries.

6.4 Security analysis of the recomposition technique

The results presented so far had no direct connection with GLV/GLS curves. We now turn to such curves, and first discuss in this section the security of what we called the "recomposition technique" for GLV/GLS coefficients (namely, choose k_1 and k_2 uniformly at random in some interval $[0, K)$ to obtain $k = k_1 + k_2\lambda \bmod n$), whereas the next section will focus on the "decomposition technique".

To fix ideas, we consider an elliptic curve E obtained by the quadratic GLS method over a prime field [GLS11, §2.1]. In other words, there is an elliptic curve E_0 over the prime field $\mathbb{F}_p$ such that E is the quadratic twist of E_0 over $\mathbb{F}_{p^2}$. If we denote by $p + 1 - t$ the order of $E_0(\mathbb{F}_p)$ (where t is bounded as $|t| \leq 2\sqrt{p}$ by the Hasse–Weil theorem), the order n of $E(\mathbb{F}_{p^2})$ satisfies:

$$n = (p-1)^2 + t^2. \tag{6.1}$$

We assume that this order n is prime, which is the main case of interest. Then, E is endowed with an efficient endomorphism ψ (obtained by conjugating the Frobenius map with the twisting isomorphism) which acts on the cyclic group $E(\mathbb{F}_{p^2})$ by multiplication by

$$\lambda \equiv t^{-1}(p-1) \pmod{n}. \tag{6.2}$$

In particular, $\lambda^2 \equiv (p-1)^2/t^2 \equiv -t^2/t^2 \equiv -1 \pmod{n}$.

In this setting, we first prove in §6.4.1 that if k_1 and k_2 are chosen uniformly at random in $[0, \sqrt{n})$, then $k = k_1 + k_2\lambda$ is statistically close to uniform in $\mathbb{Z}/n\mathbb{Z}$, so that such a choice of (k_1, k_2) can be used securely in any cryptographic protocol (and in particular ECDSA). On the other hand, we show in §6.4.2 that if k_1 and k_2 are chosen in $[0, 2^m)$ where $m = \lfloor \frac{1}{2} \log_2 n \rfloor$ instead, then $k = k_1 + k_2\lambda$ may not be close to uniform anymore, and we show that a variant of Bleichenbacher's attack can apply. In §6.4.3, we describe an implementation of that attack on a 160-bit GLS curve, similar to the attack of §6.3.

6.4.1 A secure choice of (k_1, k_2)

Let E be a curve of prime order n over $\mathbb{F}_{p^2}$ obtained by the quadratic GLS method as above. In view of (6.1), we have:

$$(p-1)^2 \leq n \leq (p-1)^2 + (2\sqrt{p})^2 = (p+1)^2,$$

and the inequalities are in fact strict, since n is prime. Thus, we have $p-1 < \sqrt{n} < p+1$, and it follows that the distribution of $k = k_1 + k_2\lambda$ for (k_1, k_2) uniform in $[0, \sqrt{n})^2$ is statistically close to the distribution of the same k for (k_1, k_2) uniform in $[0, p-1)^2$. We will thus concentrate on the latter, and show that it is close to uniform in $\mathbb{Z}/n\mathbb{Z}$ using the following lemma.

Lemma 6.3. *The following map is injective.*

$$\begin{aligned} F\colon [0, p-1)^2 &\longrightarrow \mathbb{Z}/n\mathbb{Z} \\ (k_1, k_2) &\longmapsto k_1 + k_2\lambda. \end{aligned}$$

Proof. Consider two distinct pairs $(k_1, k_2) \neq (k_1', k_2')$ such that $F(k_1, k_2) = F(k_1', k_2')$. We have:

$$\begin{aligned} (x - x') + (y - y')\lambda &\equiv 0 \pmod{n} \\ (x - x')^2 &\equiv \lambda^2 (y - y')^2 \pmod{n} \\ (x - x')^2 + (y - y')^2 &\equiv 0 \pmod{n}, \end{aligned}$$

since $\lambda^2 \equiv -1 \pmod{n}$. Thus, the positive integer $(x - x')^2 + (y - y')^2$ is divisible by n, and it is also smaller than $2(p-1)^2 < 2n$, so we must have $(x - x')^2 + (y - y')^2 = n$. In other words, $(x - x')^2 + (y - y')^2$ is a decomposition of n as a sum of two squares. Now it is well-known that, as a prime number, n has at most one decomposition as a sum of two squares up to order and sign (see e.g. [MM99, §3.6]), and $(p-1)^2 + t^2$ is one such representation. As a result, we must have either $x - x' = \pm(p-1)$ or $y - y' = \pm(p-1)$, and neither is possible since those difference are bounded by $p-2$ in absolute value. Hence, F is injective as required. ∎

Theorem 6.1. *The distribution of the values $k = k_1 + k_2\lambda$ for (k_1, k_2) uniform in $[0, p-1)^2$ is statistically close to the uniform distribution on $\mathbb{Z}/n\mathbb{Z}$. More precisely, the statistical distance:*

$$\Delta_1 = \sum_{k \in \mathbb{Z}/n\mathbb{Z}} \left| \Pr\left[k = k_1 + k_2\lambda \ ; \ (k_1, k_2) \xleftarrow{\$} [0, p-1)^2\right] - \frac{1}{n} \right|$$

is given by $\Delta_1 = 2t^2/n$, which is negligible.

Proof. Indeed, since the function F above is injective by Lemma 6.3, the probability $\Pr\left[k = k_1 + k_2\lambda \ ; \ (k_1, k_2) \xleftarrow{\$} [0, p-1)^2\right]$ is equal to $1/(p-1)^2$ for each of the $(p-1)^2$ points in the image of F, and 0 for each of the $n - (p-1)^2 = t^2$ points outside of that image. Therefore:

$$\Delta_1 = (p-1)^2 \cdot \left| \frac{1}{(p-1)^2} - \frac{1}{n} \right| + t^2 \cdot \left| 0 - \frac{1}{n} \right| = 1 - \frac{(p-1)^2}{n} + \frac{t^2}{n} = \frac{2t^2}{n}$$

as required. This is bounded above by $8p/(p-1)^2$, which is indeed negligible. ∎

Remark. Theorem 6.1 means that it is secure, in any ECC protocol instantiated over the GLS curve E, to sample random scalars k by picking k_1 and k_2 uniformly in $[0, p-1)$, or equivalently $[0, \sqrt{n})$.

As we can see, the proof relies on the particular arithmetic properties of the quadratic GLS method (mainly the fact that $\lambda = \sqrt{-1}$ in $\mathbb{Z}/n\mathbb{Z}$), so that the result does not readily extend to different settings, like the GLV method on a curve of CM discriminant -3. And indeed, in that case, Brumley and Nyberg have provided evidence that choosing (k_1, k_2)

uniformly in $[0, \sqrt{n})$ may not yield a close to uniform distribution for k [BN09, Example 3]. They suggest an alternate approach to select intervals to choose k_1 and k_2 from and still achieve high entropy in a more general setting, but since the quadratic GLS method is one of the most used variants of GLV/GLS, we believe Theorem 6.1 is of significant practical interest.

6.4.2 Breaking insecure choices of (k_1, k_2) with Bleichenbacher's attack

In the quadratic GLS setting, we have just seen that choosing (k_1, k_2) uniformly in $[0, \sqrt{n})^2$ yields a close-to-uniform distribution of $k = k_1 + k_2\lambda$. However, we can reasonably suspect that if we choose k_1 and k_2 uniformly in $[0, 2^m)$, $m = \lfloor \frac{1}{2} \log_2 n \rfloor$ (i.e. uniform bitstrings of length just under half of the size of n), the distribution of k will no longer be uniform. This is not immediately visible on the bias, however.

Indeed, if we let $T = 2^m$ and define K_1, K_2 as independent uniform random variables over $[0, T)$ and K as the random variable in $\mathbb{Z}/n\mathbb{Z}$ given by $K = K_1 + K_2\lambda$, we have, by Lemma 6.1:

$$B_n(K) = B_n(K_1) \cdot B_n(\lambda K_2) = \frac{1}{T}\left|\frac{\sin(\pi T/n)}{\sin(\pi/n)}\right| \cdot \frac{1}{T}\left|\frac{\sin(\pi\lambda T/n)}{\sin(\pi\lambda/n)}\right|.$$

The first factor is very close to 1, but the second factor is usually negligible. For example, on the 160-bit GLS curve (6.3) below, we have $T = 2^{79}$ and $B_n(\lambda K_2) \approx 1.52/T$. As a result, Bleichenbacher's attack does not apply directly to this setting in general.

However, since $\lambda \equiv t^{-1}(p-1) \pmod{n}$, we claim that there is a significant bias on the values $t \cdot k$. Indeed, we have:

$$\begin{aligned}
B_n(tK) &= B_n(tK_1) \cdot B_n\big((p-1)K_2\big) \\
&= \frac{1}{T}\left|\frac{\sin(\pi t T/n)}{\sin(\pi t/n)}\right| \cdot \frac{1}{T}\left|\frac{\sin(\pi(p-1)T/n)}{\sin(\pi(p-1)/n)}\right| \\
&= \frac{1}{T}\frac{\pi t T/n + O((tT/n)^3)}{\pi t/n + O((t/n)^3)} \cdot \frac{1}{T}\frac{|\sin(\pi(p-1)T/n)|}{\pi(p-1)/n + O(((p-1)/n)^3)} \\
&= \Big(1 + O\big((tT/n)^2 + (p/n)^2\big)\Big) \cdot \left|\frac{\sin(\pi(p-1)T/n)}{\pi(p-1)T/n}\right|.
\end{aligned}$$

The big-O in the first factor is negligible since $tT/n = \Theta(p^{1/2} \cdot p/p^2) = \Theta(p^{-1/2})$ and $p/n = \Theta(p^{-1})$. On the other hand, $(p-1)T/n \approx T/\sqrt{n}$ is roughly between 0.5 and 1 depending on how close n is to a power of two. Thus, the bias is significant in general, and is maximal when $(p-1)T/n$ is smallest (close to 1/2), which happens when n is just under a power of two. The bias $B_n(tK)$ is then close to $1/(\pi \cdot 1/2) = 2/\pi \approx 0.637$.

It is then straightforward to adapt Bleichenbacher's attack to this setting by targetting the values $t \cdot k$ instead of k. We can then break ECDSA signatures that use nonces of the form $k = k_1 + k_2\lambda$ above using that variant. An implementation of that attack is discussed in the next subsection.

6.4.3 Implementation of Bleichenbacher's attack in the GLS setting

We carry out the attack described above on the 160-bit GLS curve E defined as follows. Over the 80-bit prime field[1] $\mathbb{F}_p$, $p = 255 \cdot 2^{72} + 1$, we define $E_0 \colon y^2 = x^3 - 3x/23 + 104$. Then, the elliptic curve E is the quadratic twist of E_0 over $\mathbb{F}_{p^2} = \mathbb{F}_p(\sqrt{23})$, namely:

$$E \colon y^2 = x^3 - 3x + 104 \cdot \sqrt{23}^3 \text{ over } \mathbb{F}_{p^2}. \tag{6.3}$$

[1] This is an example of "optimal prime field" (OPF). See e.g. [WG12].

The order of $E_0(\mathbb{F}_p)$ is $p+1-t$ for $t = 776009485427$, and $E(\mathbb{F}_{p^2})$ is of prime order $n = (p-1)^2 + t^2$. The theoretical value of the bias $B_n(tK)$, computed using the exact formula above, is then ≈ 0.634.

We performed the recovery of 32 most significant bits of a private key as in section 6.3.2. We computed 2^{33} signatures and unrolled the attack on $(tc_j \mod n, th_j \mod n)$ instead of (c_j, h_j). We checked the bias and obtained ≈ 0.634116 which is close to the theory. In practice the attack took about 2^{24} CPU-seconds, with 56% for the signature generation, 37% for the four sort-and-difference reduction steps, 5% for the candidate selection and FFT table preparation and less than 0.5% for the FFT itself. In wall-clock time terms, except for the signature generation which took (much) longer, other phases were identical as 6.3.2. We attribute this unexpected increase in signing time to threshold effects: for example, representing elements on a prime field with $\approx 2^{160}$ elements needs only 3 64-bit words, whereas a on $\mathbb{F}_{p^2}$ we needed $4 * 2 = 8$ words.

6.5 Security analysis of the decomposition technique

In this section, we analyze the security of algorithms for computing the decomposition of the nonces used in the GLV method from a side-channel analytic perspective. Many techniques have been proposed, including [GLV01, PJKL02]. The original GLV method [GLV01] based on LLL reduction of a lattice that depends on the nonce k, and variants thereof, have an execution time that depends on k, and are therefore vulnerable to timing attacks.

Therefore, we examine the security of a potentially more secure approach, the Park et al. [PJKL02] decomposition technique, using more involved power analysis technique.

6.5.1 Decomposition Algorithm

Park et al. provide an alternative decomposition to the GLV paper [GLV01] which reduces the theoretical bound for the decomposition using the theory of μ-Euclidian algorithm and is a little bit faster. The algorithm requires two short and independent vectors v_1 and v_2 of the two-dimensional lattice $L = \{(x, y) : x + y\lambda = 0 \bmod n\}$. We can find these vectors during a precomputation time using the Gauss reduction. The algorithm consists in finding a vector in the lattice $L = \mathbb{Z}v_1 + \mathbb{Z}v_2$ that is close to $(k, 0)$ using linear algebra. Then, (k_1, k_2) is determined by the equation:

$$(k_1, k_2) = (k, 0) - (\lfloor b_1 \rceil v_1 + \lfloor b_2 \rceil v_2),$$

where $(k, 0) = b_1 v_1 + b_2 v_2$ is an element of $\mathbb{Q} \times \mathbb{Q}$.

Algorithm 6.4 Decomposition technique of Park et al. in [PJKL02].

Require: $k \approx n$, the shortest vectors $v_1 = (x_1, y_1), v_2 = (x_2, y_2)$
Ensure: (k_1, k_2) such that $k = k_1 + k_2\lambda (\mathrm{mod} n)$

1: $D = x_1 y_2 - x_2 y_1, a_1 = y_2 k, a_2 = -y_1 k$
2: $z_i = \lfloor a_i / D \rceil$ for $i = 1, 2$
3: $k_1 = k - (z_1 x_1 + z_2 x_2)$, $k_2 = z_1 y_1 + z_2 y_2$ **return** (k_1, k_2)

The decomposition technique depicted in Algorithm 6.4 makes many computations involving the sensitive nonce k. Particularly, the computation of a_1 (*resp.* a_2) is based on a multiplication of the nonce k by y_2 (*resp.* y_1) which is assumed to be known since it is a precomputed value obtained from public parameters using a deterministic algorithm.

Suppose now that we obtain the knowledge of the least significant byte of ℓ nonces $k_1, \cdots, k_\ell$. The best strategy for finding the secret key x consists in performing classical lattice attacks as proposed in [HGS01, NS02, NS03]. For a 160-bit modulus, the lattice attack works consistently for $\ell \gtrsim 27$. However the side-channel attack may sometimes fail, *i.e.* the returned byte of some k_j can be a wrong value. Thus, by denoting $0 < c < 1$ the confidence rate, the side-channel attack has to be performed on $m > \lceil 27/c \rceil$ signatures. Then:

- Select 27 signatures at random among them.
- Perform the attack using these signatures.
- If the attack fails, goto the first step.

The probability of success at each iteration of the lattice attack is $\binom{m \cdot c}{27} / \binom{m}{27}$. As an example, suppose we obtain $m = 200$ signatures, and can guess the least significant byte with 90% accuracy ($c = 0.9$). Then the probability of success of the lattice attack is about 4.7% and 21 lattice reductions have to be performed on average. Since LLL reductions are cheap, much lower success probabilities are tractable as well.

In the following, we discuss the side-channel attack that aims at recovering the first byte of the nonce targeting the two aforementioned multiplications. We present the attack in the particular case of a 8-bit implementation (that corresponds to the device we used in experiments). Note that this attack may also work for 16-bit implementation but in this case the computational cost will be larger and the success rate smaller.

6.5.2 Side-Channel Attack on this implementation

The details of the attack highly rely on the way the multiplication is implemented. Depending of the underlying algorithm, the attack may be more or less difficult. We present here the attack corresponding to the implementation we target but we will discuss adaptations to different algorithms. The multiplication we target is a schoolbook multiplication with the nonce being scanned in the outter loop. Algorithm 6.5 outlines the implementation of such multiplication for ℓ_n-bit nonces and $\ell_{n/2}$-bit b.

Algorithm 6.5 Multiplication $v = kb$ of $k = \sum_{i=0}^{\ell_n/8} k_i 2^{8i}$ times $b = \sum_{i=0}^{\ell_{n/2}/8} b_i 2^{8i}$.

Require: ℓ_n-bit k and $\ell_{n/2}$-bit b two integers, $v = 0$
Ensure: $v = k \times b$

1: $v \leftarrow 0$
2: **for** $i = 0$ to $i < \ell_n/8$ **do**
3: $\quad c_0 \leftarrow 0$
4: $\quad$ **for** $j = 0$ to $j < \ell_{n/2}/8$ **do**
5: $\quad\quad v_{i+j} = (k_i \times b_j + c_j)$ & `0xFF`
6: $\quad\quad c_{j+1} = (k_i \times b_j + c_j) \gg 8$
7: $\quad$ **end for**
8: **end for** **return** v

The idea is to take profit of all operations involving the first nonce-byte in the inner loop to recover its value. This can be done by propagating a probability distibution from an operation to another and updating it with the corresponding leakages. Since the nonce

bits have to be recovered using a single trace (the nonce is randomly generated for each signature) we place ourselves in the context of a profiled attack. The application of such an attack in a non-profiled setting is left as an open question.

Template Attack on One Step. One step of the inner loop consists in a multiplication of the first byte of the nonce k_0, a byte of the auxiliary input b_i and the carry c_i. This results in a value v_i and a new carry c_{i+1}. We may obtain leakages for each of these variables. We denote by capital letters the output distributions of template exploitation corresponding to small letter variables. For instance, after processing the leakage corresponding to c_i, the attacker gets a distribution

$$C_i = \big(\Pr(c_i = 0), \Pr(c_i = 1), \ldots, \Pr(c_i = 255)\big).$$

Since these variables may be manipulated more than once during the computation, different leakage points may be combined by multiplicating probabilities then normalizing the resulting distribution. More precisely, let $l_1, l_2, \ldots, l_l$ be leakages corresponding to variable k_0, then the distribution K_0 obtained from these leakages is computed as

$$\Pr(k_0 = x) = \frac{1}{Z} \prod_{j=1}^{l} \Pr(k_0 = x | l_j),$$

where the normalizing coefficient Z is given by $\sum_x \prod_{j=1}^{l} \Pr(k_0 = x|l_j)$.

Propagating and Updating Distribution. Let us now discuss how to take profit of all the leakages of the inner loop to gain information on the byte k_0. The main idea is to gather all the information from all variables of a given step i into distribution K_0 and C_{i+1} then do the same at step $i+1$ using the newly updated distributions. From a probabilistic point of view we should compute the joint distribution of variables of step i then compute marginalized distributions K_0 and C_{i+1}. The following algorithm updates the distributions K_0 and C_{i+1} according to the distributions of variables b_i, v_i and c_i.

Algorithm 6.6 Information propagation for one step of the multiplication inner loop.

Require: distributions K_0, B_i, V_i, C_i and C_{i+1}
Ensure: K'_0 and C'_{i+1} updated distribution

1: $K'_0 = (0, 0, \ldots, 0)$
2: **for** $0 \leq k, b, c < 256$ **do**
3: $\quad 2^8 \cdot u + v \leftarrow k \times b + c$
4: $\quad K'_0(k) \leftarrow K'_0(k) + K_0(k) \cdot B_i(b) \cdot C_{i-1}(c) \cdot V_i(v) \cdot C_i(u)$
5: $\quad C'_{i+1}(u) \leftarrow C'_{i+1}(u) + K_0(k) \cdot B_i(b) \cdot C_i(c) \cdot V_i(v) \cdot C_{i+1}(u)$
6: **end for**
7: **return** $K'_0 / \sum_k K'_0(k)$ and $C'_{i+1} / \sum_u C'_{i+1}(u)$

The attacker starts with using Algorithm 6.6 for the first step. Then she uses the newly updated distributions K_0 and C_1 and the initial distributions B_1, V_1 and C_2 as inputs of Algorithm 6.6 and so on ... At the end, the attacker gets the final distribution K_0 from which she can derive the most likely value of the least significant bit (or more).

6.6 Automatically finding fault attacks

We refer to Chapter 4 for the definition of fault conditions and the technical description of the tool we used for automatically finding fault attacks.

6.6.1 Fault conditions for ECDSA signatures

Compared to the fault conditions for RSA signatures, those we consider here are of a different nature, that rely on partial knowledge of the nonce k used in the computation. We first consider a novel fault condition focusing only on faulting the scalar multiplication. Then, we discuss an already-exploited fault condition where k can be faulted during both the scalar multiplication and the computation of its inverse, as considered in [NNTW05].

In both cases, knowing some bits of the nonce k is sufficient to mount a classic lattice-based attack.

In the following, we assume that the message to be signed is known and its hash value is h and we denote $\mathtt{abs}$ the abscissa of an elliptic curve point, $\text{LSB}_\ell k$ the ℓ least significant bits of k and $\gg$ for the right-shift operator.

Faulting r

Our fault condition considers faulted signatures such that r is computed using only some of the bits of k:

Proposition 6.2. *Given sufficiently many values satisfying one of the fault conditions:*

$$r, s : \exists k.\ r = \mathtt{abs}([k \gg \ell] \cdot P) \wedge s = k^{-1}(h + rx) \bmod q \tag{6.4}$$

$$r, s : \exists k.\ r = \mathtt{abs}(\pm[2^\ell] \cdot [k \gg \ell] \cdot P) \wedge s = k^{-1}(h + rx) \bmod q \tag{6.5}$$

one can efficiently retrieve the secret key x.

The proof of this proposition can be done in two parts, summed up by two facts. We do the proof for condition (6.4). It should be clear that the same proof applies for condition (6.5). In particular, Fact 6.2 tells us that it is sufficient to be able to recover ℓ bits of k to recover the secret key x, and the proof of Fact 6.1 clearly generalizes to condition (6.5), since its proof revolves around computations on curve points $\pm[2^\ell] \cdot [k \gg \ell] \cdot P$.

Fact 6.1. *Given a single pair (r, s) that satisfies the fault condition:*

$$r, s : \exists k.\ r = \mathtt{abs}([k \gg \ell] \cdot P) \wedge s = k^{-1}(h + rx) \bmod q,$$

one can efficiently retrieve the ℓ least significant bits of k.

Proof. Indeed we have the following equalities:

$$[\text{LSB}_\ell k] \cdot P = [k] \cdot P - [2^\ell] \cdot [k \gg \ell] \cdot P,$$
$$[k] \cdot P = \left[\frac{h}{s}\right] \cdot P + \left[\frac{r}{s}\right] \cdot Q,$$

the first one results from the property on the faulted value r and the second one from the verification equation of ECDSA. The second equality implies that one can compute $[k] \cdot P$ using public parameters and the signatures. Our goal is to recover $\text{LSB}_\ell k$. To this end, we show that we can compute the right hand side of the first equation since the first part

is $[k] \cdot P$ and the second part is one of the points whose abscissa is r. Consequently, we can recover $\pm[2^\ell] \cdot [k \gg \ell] \cdot P$ by solving in y the quadratic equation $y^2 = r^3 + ax + b$ if the elliptic curve is the set of points (x, y) satisfying the equation $y^2 = x^3 + ax + b$. For one of these two candidates, the point has a small discrete log related to P and we can recover them using Pollard's lambda method for instance in time $O(2^{\ell/2})$. This algorithm will succeed only for the correct candidate and will likely fail for the other one.

■

Now, given sufficiently many faulty signatures, the secret x can be recovered using a technique based on lattices.

Fact 6.2. *Given a sufficient number of ECDSA signatures whose nonces k are partially known, one can efficiently retrieve the secret key x.*

The idea of the proof consists in using the congruence:

$$xr = sk - h \bmod q.$$

This relation reveals nothing about x when k is chosen truly at random in $\mathbb{Z}/q\mathbb{Z}$, but when some bits of k are known, some information about x is leaked since all other parameters are publicly known. Intuitively, knowing ℓ bits of k should reveal ℓ bits of information about x, so if q is n bits long, x should be recoverable from about n/ℓ faulty signatures. Moreover, the relation is affine in both x and k, and the problem can be attacked with lattices. Efficient algorithms for recovering x from a set of such signatures can be found in [HGS01, NS03, NT12].

Note to conclude that a similar result holds for the most significant bits. For example, condition (6.4) in this case could be written as follows.

$$r, s : r = \mathtt{abs}([k \bmod 2^{n-\ell}]P) \wedge s = k^{-1}(h + rx) \bmod q.$$

In this case, we retrieve the most significant bits of the nonces and it is not difficult to adapt the lattice to this case.

Using short randomness: faulting r and s

We also consider the following fault condition, implicitly used in the original attack on ECDSA by Nguyen et al. [NNTW05], where both the scalar multiplication and field inversion are faulted to simulated short values for k (that is, values whose most significant or least significant bits are zero).

Proposition 6.3 ([NNTW05])**.** *Given a sufficient number of pairs (r, s) that satisfy the fault condition:*

$$r, s : \exists k.\ r = \mathtt{abs}([lsb(k)] \cdot P) \wedge s = lsb(k)^{-1}(h + rx) \bmod q,$$

one can efficiently recover the secret key x.

Although we do not prove it, the validity of this fault condition is justified by its use in existing attacks.

Implementation and evaluation

We also implement our key recovery attacks on ECDSA in Sage to evaluate their performance. Some experimental values of (ℓ, d) are given in Table 6.3.

Table 6.3: Minimal number of signatures d to be faulted depending on ℓ using curves brainpoolP160r1, brainpoolP256r1 and brainpoolP384r1. The percentage given in one case represents the success rate of the attack and could be increased by increasing the value of d.

q	160 (bits)			256 (bits)			384 (bits)	
ℓ	4	8	16	8	16	32	8	16
d	61 ($\simeq$70%)	23	11	38	17	9	61	26

6.6.2 Results of our tool

We run our tool on the ECDSA signature algorithm. We consider an implementation where scalar multiplication is computed using MSB-first Double-and-Add (Algorithm 6.1). The main challenge here is that the fault conditions we consider are very precise, in the sense that they give a full functional description of the result depending on some (faulted) inputs. We therefore need not only to be able to find the faults, but also to be able to prove the functional correctness of the non-faulted algorithms.

Faults on the randomness

We first consider the fault condition from Proposition 6.3, that we generalized from the attack of [NNTW05]. The tool finds that performing a zero-higher-order bit fault on k after it is sampled is sufficient to guarantee the fault condition (as we then have $k = k \gg \ell$). However, we do not automatically find more complex attacks (that use Proposition 6.3) on the algorithms computing scalar multiplications and field element inversions. We believe that our tool would in fact find such attacks given precise enough implementations for these operations, and precise enough loop invariants for their non-faulted versions.

Faults on scalar multiplication

Fault condition (6.4) from Proposition 6.2 allows our algorithm to quickly focus the fault search on the computation of the scalar multiplication in ECDSA. The tool discovers that exiting the loop early when computing $[k] \cdot P$, and letting all other computations occur normally, yields signatures (r, s) that fulfill fault condition (6.4).

The second fault condition (6.5) from Proposition 6.2 leads to a slightly more flexible overall attack, since it does not require the number of faulted iterations to be known. Given this fault condition and an abstract algorithmic description of the ECDSA algorithm, our tool finds that forcing the branch condition at line 5 (Algorithm 6.1) to false for a number of iterations towards the end of the loop yields an exploitable result. Generalizing, faulting line 5 or its implementation such that it computes $R_0 \leftarrow \pm R_0$ instead of $R_0 \leftarrow R_0 + P$ would yield the same result.

Faults on point addition

This observation leads us to consider more concrete refinements of the point addition algorithm. In particular, we consider a register-level algorithm for Jacobian-Jacobian point

addition, as presented by Murdica [Mur14, Algorithm 36]. This algorithm, shown in Algorithm 6.7, is only correct when applied to distinct curve points Q, R that are not at infinity or inverse of each other. It assumes R is initially stored in registers (T_1, T_2, T_3), and Q in (T_4, T_5, T_6). Note that the values of T_4, T_5 and T_6 are untouched by the algorithm. All the faults we detail are such that these registers are left untouched by the faulted algorithm as well.

Given the implementation where the partial point addition algorithm is wrapped in tests ensuring it is applied correctly (that is, $Q, R \neq \infty$ and $Q \neq \pm R$), our tool quickly finds that faulting the conditional checks is sufficient to force the fault condition: by faulting the test that checks whether the second argument is infinite, we can easily force the wrapped addition algorithm to return its first argument, forcing the fault condition.

However, since the base point P is of order q, and R_0 is always a scalar multiple of P, such checks can be optimized away when the addition algorithm is used for scalar multiplication.

With an additional condition that none of the scalar multiples of P are on the vertical axis our tool finds null faults, and some faults in combined models involving null faults and instruction skips, that lead to the faulted computation of $R_0 + P$ returning $-R_0$. We describe three of them in Algorithm 6.7, variables marked with a symbol are set to 0 during the corresponding attack. Full lines marked with a symbol are skipped by the corresponding attack.

Performing this fault during the last iterations of the Double-and-Add loop then yields a faulted ECDSA signature that fulfills fault condition (6.5) and can be used in the lattice-based attack. Our tool yields a list of more than 100 ways to fault point addition in a useful way:

- 1 attack involving 3 null faults (marked using †);
- 116 attacks (some are variants of each other) involving 4 null faults (an example is given using ∘);
- 9 attacks involving 2 null faults and 2 instruction skips (an example is shown using ⋆).

Algorithm 6.7 Elliptic curve point addition.

1: **function** EcAdd(R,Q)
2: $T_7 = T_3 \cdot T_6$;
3: $T_8 = T_3^2\{\dagger, \circ\}$;
4: $T_3 = T_3 \cdot T_8$;
5: $T_3 = T_5 \cdot T_3$;
6: $T_9 = T_4 \cdot T_8$;
7: $T_8 = T_6^2$;
8: $T_1 = T_1 \cdot T_8$;
9: $T_8 = T_8 \cdot T_6$;
10: $T_2 = T_2 \cdot T_8$;
11: $T_9 = T_9\{\star\} - T_1$;
12: $T_8 = T_3 - T_2\{\dagger, \circ\}$;
13: $T_3 = T_7 \cdot T_9$;
14: $T_7 = T_9^2$;
15: $T_9 = T_7\{\circ\} \cdot T_9$;
16: $T_7 = T_1 \cdot T_7\{\dagger\}$;
17: $T_2 = T_2 \cdot T_9$;
18: $T_1 = T_8^2$;
19: $T_1 = T_1\{\star\} - T_9$;
20: $T_1 = T_1 - T_7; \{\star\}$
21: $T_1 = T_1 - T_7; \{\star\}$
22: $T_7 = T_7 - T_1$;
23: $T_8 = T_8 \cdot T_7$;
24: $T_2 = T_8 - T_2\{\circ\}$;
25: **return** (T_1, T_2, T_3);
26: **end function**

Chapter 7
Binary Elligator Squared

7.1 Introduction

Applications of elliptic curve cryptography to anonymity, privacy and censorship circumvention call for methods to represent uniformly random points on elliptic curves as uniformly random bit strings, so that, for example, ECC network traffic can masquerade as random traffic.

At ACM CCS 2013, Bernstein et al. proposed an efficient approach, called "Elligator", to solve this problem for arbitrary elliptic curve-based cryptographic protocols, based on the use of efficiently invertible maps to elliptic curves. Unfortunately, such invertible maps are only known to exist for certain classes of curves, excluding in particular curves of prime order and curves over binary fields. A variant of this approach, "Elligator Squared", was later proposed by Tibouchi (FC 2014) supporting not necessarily injective encodings to elliptic curves (and hence a much larger class of curves), but, although some rough efficiency estimates were provided, it was not clear how an actual implementation of that approach would perform in practice.

In this chapter, we show that Elligator Squared can indeed be implemented very efficiently with a suitable choice of curve encodings. More precisely, we consider the binary curve setting (which was not discussed in Tibouchi's paper), and implement the Elligator Squared bit string representation algorithm based on a suitably optimized version of the Shallue–van de Woestijne characteristic 2 encoding, which we show can be computed using only multiplications, trace and half-trace computations, and a few inversions.

On the fast binary curve of Oliveira et al. (CHES 2013), our implementation runs in an average of only 22850 Haswell cycles, making uniform bit string representations possible for a very reasonable overhead—much smaller even than Elligator on Edwards curves.

We also compare implementations of Elligator and Elligator Squared on a curve supported by Elligator, namely Curve25519. We find that generating a random point and its uniform bit string representation is around 35–40% faster with Elligator for protocols using a fixed base point, but 30–35% faster with Elligator Squared in the case of a variable base point. Both are significantly slower than our binary curve implementation.

This work was presented at SAC 2014 [AFQ+14].

7.1.1 Context

For censorship circumvention applications, however, ECC presents a weakness: points on a given elliptic curve, when represented in a usual way (even in compressed form) are easy to distinguish from random bit strings. For example, the usual compressed bit string representation of an elliptic curve point is essentially the x-coordinate of the point, and only about half of all possible x-coordinates correspond to valid points (the other half being x-coordinates of points of the quadratic twist). This makes it relatively easy for an attacker to distinguish ECC traffic (the transcripts of multiple ECDH key exchanges, say)

from random traffic, and then proceed to intercept, block or otherwise tamper with such traffic.

To alleviate that problem, one possible approach is to modify protocols so that transmitted points randomly lie either on the given elliptic curve or on its quadratic twist (and the curve parameters must therefore be chosen to be twist-secure). This is the approach taken by Möller [Möl04], who constructed a CCA-secure KEM with uniformly random ciphertexts using an elliptic curve and its twist. This approach has also been used in the context of kleptography, as considered by Young and Yung [YY07, YY10], and has already been deployed in circumvention tools, including StegoTorus [WWY+12], a camouflage proxy for Tor, and Telex [WWGH11], an anticensorship technology that uses a covert channel in TLS handshakes to securely communicate with friendly proxy servers. However, since protocols and security proofs have to be adapted to work on both a curve and its twist, this approach is not particularly versatile, and it imposes additional security requirements (twist-security) on the choice of curve parameters.

A different approach, called "Elligator", was presented at ACM CCS 2013 by Bernstein, Hamburg, Krasnova and Lange [BHKL13a]. Their idea is to leverage an efficiently computable, efficiently invertible algebraic function that maps the integer interval $S = \{0, \ldots, (p-1)/2\}$, p prime, *injectively* to the group $E(\mathbb{F}_p)$ where E is an elliptic curve over $\mathbb{F}_p$. Bernstein et al. observe that, since ι is injective, a uniformly random point P in $\iota(S) \subset E(\mathbb{F}_p)$ has a uniformly random preimage $\iota^{-1}(P)$ in S, and use that observation to represent an elliptic curve point P as the bit string representation of the unique integer $\iota^{-1}(P)$ if it exists. If the prime p is close to a power of 2, a uniform point in $\iota(S)$ will have a close to uniform bit string representation.

This method has numerous advantages over Möller's twisted curve method: it is easier to adapt to existing protocols using elliptic curves, since there is no need to modify them to also deal with the quadratic twist; it avoids the need to publish a twisted curve counterpart of each public key element, hence allowing a more compact public key; and it doesn't impose additional security requirements like twist-security. But it crucially relies on the existence of an injective encoding ι, only a few examples of which are known [Far11, FJT13, BHKL13a], all of them for elliptic curves of non-prime order over large characteristic fields. This makes the method inapplicable to implementations based on curves of prime order or on binary fields, which rules out most standardized ECC parameters [FIP09, Cer10, LM10, ANS11], in particular. Moreover, the rejection sampling involved (when a point P is picked outside $\iota(S)$, the protocol has to start over) can impose a significant performance penalty.

To overcome these limitations, Tibouchi [Tib14] recently proposed a variant of Elligator, called "Elligator Squared", in which a point $P \in E(\mathbb{F}_q)$ is represented not by a preimage under an injective encoding ι, but by a randomly sampled preimage under an essentially surjective map $\mathbb{F}_q^2 \to E(\mathbb{F}_q)$ with good statistical properties, known as an *admissible encoding* following a terminology introduced by Brier et al. [BCI+10]. By results due to Farashahi et al. [FFS+13], such admissible encodings are known to exist for all isomorphism classes of elliptic curves, including curves of prime order and binary curves. Since admissible encodings are essentially surjective, the approach also eliminates the need for rejection sampling at the protocol level.

7.1.2 Our contributions

While the Elligator Squared approach is quite versatile, its efficiency is highly dependent on how fast the underlying admissible encoding can be computed and sampled, and the same can be said of Elligator in the settings where it can be used. Since, to the best of our

knowledge, no detailed implementation results or concrete performance numbers have been published so far for the underlying encodings, one only has some rough estimates to go by. For Elligator, Bernstein et al. give ballpark Westmere cycle count figures based on earlier implementation results [BHKL13b], and for Elligator Squared, Tibouchi provides some average operation counts in [Tib14] for a few selected encoding functions. No performance-oriented implementation is available for either approach.

In this chapter, we provide the first such implementation for Elligator Squared, and do so in the binary curve setting, which had not been considered by Tibouchi. Binary curves provide a major advantage for algorithms like Elligator Squared due to the existence of a point encoding function, the binary Shallue–van de Woestijne encoding [SvdW06], that can be computed without base field exponentiations. Using the framework of Farashahi et al. [FFS+13], one can obtain an admissible encoding from that function, and hence use it to implement Elligator Squared.

We propose various algorithmic improvements and computation tricks to obtain a fast evaluation of the binary Shallue–van de Woestijne encoding and of the associated Elligator Squared sampling algorithm. In particular, our description is much more efficient than the one given in [BCI+09, Appendix E].

Based on these algorithmic improvements, we performed software implementations of Elligator Squared on the record-setting binary GLS curve of Oliveira et al., defined over $\mathbb{F}_{2^{254}}$ [OLARH14]. We dedicate special attention to optimizing the performance-critical operations and introduce corresponding novel techniques, namely a new point addition formula in λ-affine coordinates and a faster approach for constant-time half-trace computation over quadratic extensions of $\mathbb{F}_{2^m}$. Moreover, timings are presented for both variable-time and constant-time field arithmetic.[1] The resulting timings compare very favorably to previously suggested estimates.

Finally, as a side contribution, we also propose concrete cycle counts on Ivy Bridge and Haswell for both Elligator and Elligator Squared on the Edwards curve Curve25519 [Ber06] based on the publicly available implementation of Ed25519 [BDL+12]. We find that, on this curve, the Elligator approach is roughly 35–40% faster than Elligator Squared for protocols that rely on fixed-base scalar multiplication, but conversely, for protocols that rely on variable-base scalar multiplication, Elligator Squared is 30–35% faster. Both approaches are significantly slower than what we achieve on the same CPU with our binary curve implementation.

7.2 Preliminaries

Let E be an elliptic curve over a finite field $\mathbb{F}_q$.

7.2.1 Well-bounded encodings

Some technical definitions are required to describe the conditions under which an "encoding function" $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ can be used in the Elligator Squared constructions. See [FFS+13, Tib14] for details.

Definition 7.1. *A function $f : \mathbb{F}_q \to E(\mathbb{F}_q)$ is said to be a B-well-distributed encoding for*

[1] We point out that using constant-time arithmetic for Elligator Squared is not required in most realistic adversarial models, but it does offer protection against very powerful distinguishing attackers, so the paranoid may prefer that option nonetheless.

a certain constant $B > 0$ if for any nontrivial character χ of $E(\mathbb{F}_q)$, the following holds:

$$\left| \sum_{u \in \mathbb{F}_q} \chi(f(u)) \right| \leq B\sqrt{q}.$$

Definition 7.2. *We call a function $f : \mathbb{F}_q \to E(\mathbb{F}_q)$ a (d, B)-well-bounded encoding, for positive constants d, B, when f is B-well-distributed and all points in $E(\mathbb{F}_q)$ have at most d preimages under f.*

7.2.2 Elligator Squared

Let $f : \mathbb{F}_q \to E(\mathbb{F}_q)$ be a (d, B)-well-bounded encoding and let $f^{\otimes 2}$ the tensor square defined by:

$$\begin{aligned} f^{\otimes 2} : \mathbb{F}_q^2 &\to E(\mathbb{F}_q) \\ (u, v) &\mapsto f(u) + f(v). \end{aligned}$$

Tibouchi shows in [Tib14] that if we sample a uniformly random preimage under $f^{\otimes 2}$ of a uniformly random point P on the curve, we get a pair $(u, v) \in \mathbb{F}_q^2$ which is statistically close to uniform. Moreover he proves that sampling uniformly random preimages under $f^{\otimes 2}$ can be done efficiently for all points $P \in E(\mathbb{F}_q)$ except possibly a negligible fraction of them [Tib14, Theorem 1]. The sampling algorithm Tibouchi proposed is described as Algorithm 7.1. The idea is to randomly pick a random u and then to compute a correct candidate v such that $P = f(u) + f(v)$. The last steps of the algorithm (step 5 to 7) are also needed in order to ensure the uniform distribution of the output (u, v).

Algorithm 7.1 Preimage sampling algorithm for $f^{\otimes 2}$.

1: **function** SAMPLEPREIMAGE(P)
2: **repeat**
3: $u \xleftarrow{\$} \mathbb{F}_q$
4: $Q \leftarrow P - f(u)$
5: $i \leftarrow \#f^{-1}(Q)$
6: $j \xleftarrow{\$} \{1, \cdots, d\}$
7: **until** $j \leq i$
8: $\{v_1, \cdots, v_t\} \leftarrow f^{-1}(Q)$
9: **return** (u, v_j)
10: **end function**

7.2.3 Shallue–van de Woestijne in Characteristic 2

In this section, we recall the Shallue–van de Woestijne algorithm in characteristic 2 [SvdW06], following the more explicit presentation given in [BCI+09, Appendix E]. An elliptic curve over a field $\mathbb{F}_{2^n}$ is a set of points $(x, y) \in (\mathbb{F}_{2^n})^2$ verifying the equation:

$$E_{a,b} : Y^2 + X \cdot Y = X^3 + a \cdot X^2 + b$$

where $a, b \in (\mathbb{F}_{2^n})^2$. Let g be the rational function $x \mapsto x^{-2} \cdot (x^3 + a \cdot x^2 + b)$. Letting $Z = Y/X$, the equation for $E_{a,b}$ can be rewritten as $Z^2 + Z = g(X)$.

Theorem 7.1. *Let* $g(x) = x^{-2} \cdot (x^3 + a \cdot x^2 + b)$ *where* $a, b \in (\mathbb{F}_{2^n})^2$. *Let*

$$X_1(t,u) = \frac{t \cdot c}{1 + t + t^2} \qquad X_2(t,u) = t \cdot X_1(t,u) + c \qquad X_3(t,u) = \frac{X_1(t,u) \cdot X_2(t,u)}{X_1(t,u) + X_2(t,u)}$$

where $c = a + u + u^2$. *Then* $g(X_1(t,u)) + g(X_2(t,u)) + g(X_3(t,u)) \in h(\mathbb{F}_{2^n})$ *where* h *is the map* $h : z \mapsto z^2 + z$.

From Theorem 7.1, we have that at least one of the $g(X_i(t,u))$ must be in $h(\mathbb{F}_{2^n})$, which leads to a point in $E_{a,b}(\mathbb{F}_{2^n})$. Indeed, we have that $h(\mathbb{F}_{2^n}) = \{z \in \mathbb{F}_{2^n} \mid \mathrm{Tr}(z) = 0\}$, where Tr is the trace operator $\mathrm{Tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2$ with:

$$\mathrm{Tr}\, z = \sum_{i=0}^{n-1} z^{2^i}$$

(one inclusion is obvious and the other one follows from the fact that the kernel of the $\mathbb{F}_2$-linear map h is $\{0,1\}$, hence its image is a hyperplane). As a result, $\sum_{i=1}^{3} \mathrm{Tr}(g(X_i)) = 0$ and therefore at least one of the X_i must satisfy $\mathrm{Tr}(g(X_i)) = 0$ since Tr is $\mathbb{F}_2$-valued. Such an X_i is indeed the abscissa of a point in $E_{a,b}(\mathbb{F}_{2^n})$, and we can find its y-coordinate by solving the quadratic equation $Z^2 + Z = g(X_i)$. That equation is $\mathbb{F}_2$-linear, so finding Z amounts to solve a linear system over $\mathbb{F}_2$. This yields the point-encoding function described in Algorithm 7.2.

In the description of that algorithm, the solution of the quadratic equation is expressed in terms of a linear map $\mathrm{QS} : \mathrm{Ker}(\mathrm{Tr}) \to \mathbb{F}_{2^n}$ ("quadratic solver"), which is a right inverse of $z \mapsto z^2 + z$. It is chosen among such right inverses in such a way that membership in its image is computed efficiently using a single trace computation. For example, when n is odd, it is customary to choose $\mathrm{QS}(x)$ as the trace zero solution of $z^2 + z = x$, in which case QS is simply the half-trace map HTr defined as:

$$\mathrm{HTr} : z \mapsto \sum_{i=0}^{(n-1)/2} z^{2^{2i}}.$$

When $n = 2m$ with m odd, we have $\mathbb{F}_{2^n} = \mathbb{F}_{2^m}[w]/(w^2 + w + 1)$ and we can define $\mathrm{QS}(x)$ as the solution $z = z_0 + z_1 w$ of $z^2 + z = x$ such that $\mathrm{Tr}(z_0) = 0$ (and this clearly generalizes to extension degrees with higher 2-adic valuation). The efficient computation of QS in that case is discussed in §7.4.

Algorithm 7.2 Shallue–van de Woestijne algorithm in characteristic 2.

Require: $a, b \in \mathbb{F}_{2^n}$ and $t, u \in \mathbb{F}_{2^n}$
Ensure: $(x, y) \in E_{a,b}$
1: $c \leftarrow a + u + u^2$
2: $X_1 \leftarrow t \cdot c/(1 + t + t^2)$
3: $X_2 \leftarrow t \cdot X_1 + c$
4: $X_3 \leftarrow X_1 \cdot X_2/(X_1 + X_2)$
5: **for** $j = 1$ to 3 **do**
6: $\quad h_j \leftarrow (X_j^3 + a \cdot X_j^2 + b)/X_j^2$
7: $\quad$ **if** $\mathrm{Tr}(h_j) = 0$ **then return** $(X_j, \mathrm{QS}(h_j) \cdot X_j)$
8: $\quad$ **end if**
9: **end for**

Algorithm 7.2 actually maps two parameters t, u to a rational point on the curve $E_{a,b}$. One can obtain a map $f\colon \mathbb{F}_q \to E_{a,b}(\mathbb{F}_q)$ by picking one of the two parameters as a suitable constant and letting the other one vary. In what follows, for efficiency reasons, we fix t and use u as the variable parameter.

One can check that the resulting function is well-bounded in the sense of §7.2.1. Indeed, the framework of Farashahi et al. [FFS+13] can be used to establish that it is a well-distributed encoding: the proof is easily adapted from the one given in [FT12] for the odd characteristic version of the Shallue–van de Woestijne algorithm. Moreover, each curve point has at most 6 preimages under the corresponding function: there are at most two values of u that yield a given value of X_1, and similarly for X_2, X_3. Thus, we obtain a (d, B)-well-bounded encoding for an explicitly computable constant B and $d = 6$.

7.2.4 Lambda affine coordinates

In order to have more efficient binary elliptic curve arithmetic, we will use lambda coordinates [Knu99, Sch00b, OLARH14]. Given a point $P = (x, y) \in E_{a,b}(\mathbb{F}_{2^n})$, with $x \neq 0$, its λ-affine representation of P is defined as (x, λ) where $\lambda = x + y/x$. The λ-affine equation of the Weierstrass Equation of the curve $y^2 + xy = x^3 + ax^2 + b$ is $(\lambda^2 + \lambda + a)x^2 = x^4 + b$. Note that the condition $x \neq 0$ is not restrictive in practice since the only point $x = 0$ satisfying Weierstrass equation is $(0, \sqrt{b})$.

7.3 Algorithmic aspects

We focus on Algorithm 7.1 proposed by Tibouchi in [Tib14], which we adapt for the specific characteristic 2 finite field. More precisely, we consider an elliptic curve over a field $\mathbb{F}_{2^n}$ that satisfies the equation in λ-coordinates:

$$E_{a,b} : (\lambda^2 + \lambda + a)x^2 = x^4 + b$$

where $a, b \in (\mathbb{F}_{2^n})^2$. The $(6, B)$-well-bounded encoding we consider for our efficient Elligator Squared implementation is the binary Shallue–van de Woestijne algorithm recalled in §7.2.3.

One of its properties is that among three candidates denoted X_1, X_2, X_3, either exactly one of them or all three are x-coordinate of a rational point over the binary elliptic curve $E_{a,b}$, and the algorithm outputs the first correct one. Owing to this property, some additional verifications are needed during preimage computation, since it is not always true that $\mathrm{SWChar2}_X(\mathrm{SWChar2}_X^{-1}(X_i)) = X_i$ for $i = 2, 3$ when it is true for $i = 1$, where we denote by $\mathrm{SWChar2}_X$ the x-coordinate of the binary Shallue–van de Woestijne algorithm, and by $\mathrm{SWChar2}_X^{-1}$ an arbitrary preimage thereof (see the discussion on the subroutine PreimagesSW in §7.3.2 for more details). We also have to consider another property of this algorithm, concerning the output. Indeed the y-coordinate has a specific form and thus, before searching for some preimages of the point Q, one has to test whether this property is verified (see the discussion on the overall complexity in §7.3.3 for more details).

The details of our preimage sampling algorithm in characteristic 2 are described in Algorithm 7.3 with t fixed to a constant such that $t(t + 1)(t^2 + t + 1) \neq 0$, i.e. $t \notin \mathbb{F}_4$. Note that we make the choice to use the λ-coordinates for efficiency reasons justified in §7.3.2. The rest of the section consists in describing the two subroutines SWChar2 and PreimagesSW, as well as in evaluating the overall complexity of Algorithm 7.3.

Algorithm 7.3 Preimage Sampling Algorithm in Characteristic 2

Precomputed: $t_1 = \frac{t}{1+t+t^2}, t_2 = \frac{1+t}{1+t+t^2}, t_3 = \frac{t(1+t)}{1+t+t^2}$
function SamplePreimage($E_{a,b}, P$)
  repeat
    repeat
      $u \xleftarrow{\$} \mathbb{F}_{2^n}$
      $R \leftarrow \text{SWChar2}(E_{a,b}, u, t_1, t_2, t_3)$
      $Q \leftarrow P - R$
    until $\lambda_Q + x_Q \in \text{Im(QS)}$ ▷ Test fails by convention for Q at infinity
    $k, S = \{v_1, \cdots, v_k\} \leftarrow \text{PreimagesSW}(E_{a,b}, Q, t_1, t_2, t_3)$
    $j \xleftarrow{\$} \{1, \cdots, 6\}$
  until $j \leq k$
  return (u, v_j)
end function

7.3.1 The subroutine SWChar2

The first subroutine represents the binary Shallue–van de Woestijne algorithm and its pseudocode for our case is given as Algorithm 7.4. Given a value $u \in \mathbb{F}_{2^n}$, it outputs the lambda coordinates of a point over the binary elliptic curve $E_{a,b}$.

Algorithm 7.4 Efficient Binary Shallue–van de Woestijne Algorithm

function SWChar2($E_{a,b}, u, t_1, t_2, t_3$)
  $c \leftarrow u^2 + u + a$
  $c_{-1} \leftarrow 1/c$
  for $j = 1$ to 3 **do** ▷ Compute h_j and perform a trace test
    $X_j \leftarrow t_j \cdot c$ ▷ or $X_3 \leftarrow X_1 + X_2 + c$
    $X_{-j} \leftarrow 1/t_j \cdot c_{-1}$ ▷ $1/t_j$ can also be precomputed
    $h_j \leftarrow (X_{-j})^2 \cdot b + X_j + a$
    if $\text{Tr}(h_j) = 0$ **then** ▷ At least one of the three potential tests will succeed
      $x \leftarrow X_j$
      $\lambda \leftarrow \text{QS}(h_j) + x$
      break ▷ Only take into account the first correct solution
    end if
  end for
  return (x, λ) ▷ Lambda coordinates of a point over $E_{a,b}$
end function

Since the field inversion is by far the most expensive field operation (see [OLARH14] for experimental timings and Table 7.2 below), we have modified Algorithm 7.2 so that we have a single inversion of c to perform. Indeed Algorithm 7.2 requires at most 4 field inversions: the first one at step 4 and the three others at step 6. However the parameters X_i and $1/X_i$ for $j = 1, 2, 3$ can be expressed using c, $1/c$ and some constants depending on t which can be precomputed (see Table 7.1). Note that X_3 can be computed as $c \cdot t_3$, or more efficiently as $X_1 + X_2 + c$ but this requires to keep in memory X_1 and X_2. Finally this algorithm requires a single field inversion, a QS computation and some negligible field operations (multiplications, squarings and trace computations).

Table 7.1: Efficient computation of values X_i and $1/X_i$ for $i = 1, \cdots 3$. The values $t_1 = \frac{t}{1+t+t^2}, 1/t_1, t_2 = \frac{1+t}{1+t+t^2}, 1/t_2$ and $1/t_3 = \frac{1+t+t^2}{t(1+t)}$ can be precomputed, with t a constant such that $t \notin \mathbb{F}_4$.

$X_1 \leftarrow t_1 \cdot c$	$X_2 \leftarrow t_2 \cdot c$	$X_3 \leftarrow X_1 + X_2 + c$
$1/X_1 \leftarrow 1/t_1 \cdot 1/c$	$1/X_2 \leftarrow 1/t_2 \cdot 1/c$	$1/X_3 \leftarrow 1/t_3 \cdot 1/c$

7.3.2 The subroutine PreimagesSW

The second subroutine is useful to compute the number of preimages of the point $Q = (x_Q, \lambda_Q)$ by Algorithm 7.4. Its pseudocode is detailed as Algorithm 7.5 and refers to the steps 5 and 8 of Algorithm 7.1.

Algorithm 7.5 Preimages Computation by Algorithm 7.4

function PreimagesSW($E_{a,b}, Q = (x_Q, \lambda_Q), t_1, t_2, t_3$)
  $k \leftarrow 0$
  $S \leftarrow \{\}$
  for $j = 1$ to 3 **do** ▷ From $x_Q = X_j(t, u)$...
    $\alpha_j \leftarrow x_Q \cdot 1/t_j + a$
    if $\mathrm{Tr}(\alpha_j) = 0$ **then** ▷ ...Test if there are some solutions
      if $j = 1$ **then** ▷ For X_1, a solution is a preimage
        $u_0 \leftarrow \mathrm{QS}(\alpha_j)$
        $u_1 \leftarrow u_0 + 1$
        $k \leftarrow 2$
        $S \leftarrow \{u_0, u_1\}$
      else ▷ For X_2, X_3, a solution is not necessarily a preimage
        $X_1 \leftarrow t_1/t_j \cdot x_Q$
        $tmp \leftarrow [(\lambda_Q + x_Q)^2 + (\lambda_Q + x_Q) + x_Q + a] \cdot (t_j/t_1)^2$ ▷ $tmp = b/X_1^2$
        $h_1 \leftarrow tmp + X_1 + a$
        if $\mathrm{Tr}(h_1) \neq 0$ **then** ▷ Test if X_1 would also be a correct x-coordinate
          $u_0 \leftarrow \mathrm{QS}(\alpha_j)$
          $u_1 \leftarrow u_0 + 1$
          $k \leftarrow k + 2$
          $S \leftarrow S \cup \{u_0, u_1\}$
        end if
      end if
    end if
  end for
  return k, S ▷ k: number of preimages, S: set of preimages
end function

This subroutine is more complex due to the properties of the Shallue–van de Woestijne algorithm. More precisely, there is an order relation in Algorithm 7.4: if X_1 corresponds to a x-coordinate of a point over the elliptic curve, then it will output this point, even if X_2 and X_3 also correspond to a possible x-coordinate. Thus, the equality $\mathrm{SWChar2}(\mathrm{SWChar2}^{-1}(X_j)) = X_j$ is true for $j = 1$ but not necessarily for $j = 2, 3$. In others words, for $j = 2, 3$ a solution of $\mathrm{SWChar2}^{-1}(X_j)$ is not necessarily a preimage of

X_j by SWChar2.

Starting from the equations $x_Q = X_j(t,u) = c(u)\cdot t_j$ for $j = 1,2,3$, with $c(u) = u^2+u+a$, the main idea of Algorithm 7.5 consists in testing if there exists some values of u which satisfy these equations. If one finds some candidates for u, one also has to verify if they really correspond to preimages by Algorithm 7.4. From an equation $x_Q = X_j(t,u)$ we can obtain an equation $u+u^2 = x_Q/t_j+a = \alpha_j(a,t)$ which has two solutions if $\mathrm{Tr}(\alpha_j(a,t)) = 0$ and no solution otherwise. As an example $\alpha_1(a,t)$ is equal to $x_Q\cdot(1+t+t^2)/t+a$. The solutions are then $u_0^1 = \mathrm{QS}(\alpha_j(a,t))$ and $u_1^1 = u_0^1+1$. There are thus at most 6 possible solutions for all values of j. Now for the cases $x_Q = X_2(t,u)$ and $x_Q = X_3(t,u)$, it remains to perform a verification. Actually, denoting u_0^2 one of both solutions of the equation $x_Q = X_2(t,u)$ if it exists, the computation of SWChar2(u_0^2) can result in $X_1(t,u_0^2)$ instead of $X_2(t,u_0^2)$, and this happens with probability $1/2$ which is the probability that $\mathrm{Tr}(h_1) = 0$. The same result holds for $x_Q = X_3(t,u)$, however note that if X_3 is solution but not X_1 then X_2 cannot be a solution since $\sum_{i=1}^{3}\mathrm{Tr}(g(X_i)) = 0$ according to Theorem 7.1. Thus the verification can focus only on X_1.

Naive implementation of the verification. A simple way for implementing the verification would consist in computing $\mathrm{QS}(\alpha_j(a,t))$ for $j = 2,3$ and then calling twice the subroutine SWChar2 (without the steps referring to X_2 and X_3) for testing if the test on the trace is true or not. However this would require an additional inversion per call to compute SWChar2. Moreover, with this naive implementation we have to compute the half trace before testing if the result will be a preimage.

Efficient implementation of the verification. Since the verification focuses only on X_1 as explained above, we propose an efficient way to compute b/X_1^2, which is required in order to perform the test $\mathrm{Tr}(h_1) = \mathrm{Tr}(X_1+a+b/X_1^2)$, without any field inversion. This trick is valuable when we are working in lambda coordinates. Our proposal has another advantage: we do not need to compute the solutions, i.e. $u_0 = \mathrm{QS}(\alpha_j(a,t))$ and $u_1 = u_0+1$, before to be sure that we will get two preimages. We thus save some quite expensive half trace computations.

Consider the equation:

$$x_Q = X_2 = t_2\cdot c = t_2\cdot X_1/t_1 \quad \text{with} \quad c = \mathrm{QS}(\alpha_2(a,t))^2+\mathrm{QS}(\alpha_2(a,t))+a.$$

X_1 can be expressed as $t_1/t_2\cdot x_Q$, whose computation is negligible for t_1/t_2 a precomputed value. Now starting from the equation of the elliptic curve in affine coordinates, i.e. $E_{a,b}$: $Y^2+X\cdot Y = X^3+a\cdot X^2+b$, we divide each term by X^2 and we evaluate the equation in the point Q. We then obtain:

$$\left(\frac{y_Q}{x_Q}\right)^2+\frac{y_Q}{x_Q} = x_Q+a+\frac{b}{x_Q^2},$$

and finally:

$$\frac{b}{X_1^2} = \left(\frac{t_2}{t_1}\right)^2\cdot\left[\left(\frac{y_Q}{x_Q}\right)^2+\frac{y_Q}{x_Q}+x_Q+a\right].$$

Assuming that $(t_2/t_1)^2$ is a precomputed constant, the computation of b/X_1^2 is not costly if y_Q/x_Q does not require an expensive operation. That is the case when we are working in λ-coordinates since $\lambda_Q = y_Q/x_Q+x_Q$. The same result obviously holds for the equation $x_Q = X_3$ by replacing t_2 with t_3.

To conclude, Algorithm 7.5 requires at most 3 QS computations and some negligible field operations (multiplications, squarings and trace computations).

7.3.3 Operation counts

We conclude this section by evaluating the average number of operations needed to evaluate Algorithm 7.3.

Proposition 7.1. *An evaluation of Algorithm 7.3 on uniformly random curve points requires, on average and with an error term of up to $O(2^{-n/2})$, 6 field inversions, 6 point additions, 9 quadratic solver computations and some negligible operations such as field multiplications, field squares and trace computations.*

Proof. The proof consists in evaluating the probability for exiting the two loops. First note that the output (x, λ) of Algorithm 7.4 has a specific property, namely $\lambda + x$ is in the image of QS. Since we want to retrieve the preimages of a point Q, we have to be sure that $\lambda_Q + x_Q$ is indeed in that image, which we test for by verifying whether $\mathrm{Tr}(\lambda_Q + x_Q) = 0$. Indeed, all elements of the form $\mathrm{QS}(z)$ have zero trace by definition, and the converse is true for reasons of dimensions. The success probability of this test is exactly $1/2$ since Q is a uniformly random curve point. We thus have on average 2 field inversions, 2 point additions and 2 quadratic solver computations for the internal loop (steps 4 to 8).

The complexity of the external loop demands to evaluate the probabilities for having 0, 2, 4 or 6 preimages of Q. Since all tests on the trace in Algorithm 7.5 succeed, independently, with probability $1/2 + O(2^{-n/2})$,[2] these probabilities are then, again with an error term of $O(2^{-n/2})$, $9/32$ for 0 preimage, $15/32$ for 2 preimages, $7/32$ for 4 preimages, and $1/32$ for 6 preimages. Thus, the probability for exiting the external loop is equal to $0 \cdot 9/32 + 1/3 \cdot 15/32 + 2/3 \cdot 7/32 + 1 \cdot 1/32 = 1/3$. These probabilities also hold for evaluating the average cost of an iteration of PreimagesSW in term of quadratic computations. With probability $15/32$ one such computation will be performed and so on. As a consequence, one iteration of PreimagesSW cost on average $\frac{15 \cdot 1 + 7 \cdot 2 + 1 \cdot 3}{32} = 1$ quadratic solver computation.

To sum up, Algorithm 7.3 requires on average $3 \cdot 2$ field inversions, $3 \cdot 2$ additions of points and $3 \cdot (2 + 1)$ quadratic solver computations, up to a $O(2^{-n/2})$ error term. ∎

Note that the efficiency of this algorithm can be improved further by choosing a sparse value of b and a value of t that yields sparse precomputed constants. Many of the field multiplications will then be computed faster.

7.4 Implementation aspects

Our software implementation targets modern Intel Desktop-based processors, making extensive use of the recently introduced AVX instruction set [FBJ+] accessible through compiler intrinsics. The curve choice is the GLS binary curve $(\lambda^2+\lambda+a)x^2 = x^4+b$ represented in λ-coordinates and defined over the quadratic extension $\mathbb{F}_{2^{254}}$. The extension is built by choosing the irreducible trinomial $g(w) = w^2 + w + 1$ over the base field $\mathbb{F}_{2^{127}}$ defined with the irreducible trinomial $f(z) = z^{127} + z^{63} + 1$. In this set of parameters, a field element a is represented as $a = a_0 + a_1 w$, with $a_0, a_1 \in \mathbb{F}_{2^{127}}$. For simplicity, the parameter t is chosen to be a random subfield element, allowing the computational savings by sparse multiplications described in the previous section.

[2] This can be justified rigorously using the fact that the corresponding function field extensions are pairwise linearly disjoint, exactly as in the image size computations of [FT12, §4]. For simplicity, we do not include the tedious Galois extension computations involved.

Squaring and multiplication

Field squaring closely mirrors the vector formulation proposed in [ALH10], with coefficient expansion implemented by table lookups performed through byte-shuffling instructions. The table lookups operate on registers only, allowing a very efficient constant-time implementation. Field multiplication is natively supported by the carry-less multiplier (PCLMULQDQ instruction), with the number of word multiplications reduced through application of Karatsuba formulae, as described in [TFHA+11]. Modular reduction is implemented with a shift-and-add approach, with careful choice of aligning vector word shifts on multiples of 8, to explore the faster memory alignment instructions available in the target platform.

Quadratic solver

For an odd extension degree m, the half-trace function $\mathrm{HTr} : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ is defined by $\mathrm{HTr}(c) = \sum_{i=0}^{(m-1)/2} c^{2^{2i}}$ and computes a solution $c \in \mathbb{F}_{2^m}$ to the quadratic equation $\lambda^2 + \lambda = c + \mathrm{Tr}(c)$. Let $\mathrm{Tr}' : \mathbb{F}_{2^{2m}} \to \mathbb{F}_2$ denote the trace function in a quadratic extension. The equation $\lambda^2 + \lambda = c$ can be solved for a trace zero element $c = c_0 + c_1 w \in \mathbb{F}_{2^{2m}}$ by computing two half-traces in $\mathbb{F}_{2^m}$, as described in [HKM09]. First, solve $\lambda_1^2 + \lambda_1 = c_1$ to obtain λ_1, and then solve $\lambda_0^2 + \lambda_0 = c_0 + c_1 + \lambda_1 + \mathrm{Tr}(c_0 + c_1 + \lambda_1)$ to obtain the solution $\lambda = \lambda_0 + (\lambda_1 + \mathrm{Tr}(c_0 + c_1 + \lambda_1))w$. This approach is very efficient for variable-time implementations and only requires two half-trace computations in the base field, where each half-trace computation employs a large precomputed table of $2^8 \cdot \lceil \frac{m}{8} \rceil$ field elements [OLARH14].

A more naive approach evaluates the function by alternating $m-1$ consecutive squarings and $(m-1)/2$ additions, with the advantage of taking constant-time (if squaring and addition are also constant-time, as in the case here). We derive a faster way to compute the half-trace function in constant-time over quadratic extension fields. Applying the naive approach to a quadratic extension allows a significant speedup due to the linear property of half-trace, by reducing the cost to essentially one constant-time half-trace computation over the base field. Since $\mathrm{Tr}'(c) = 0$, we have $\mathrm{Tr}(c_1) = 0$ and $\mathrm{Tr}(\lambda_1) = 0$ for the choice of λ_1 as the half-trace of c_1 as solution of $\lambda_1^2 + \lambda_1 = c_1$. This simplifies the expression above to $\lambda_0^2 + \lambda_0 = c_0 + c_1 + \lambda_1 + \mathrm{Tr}(c_0)$. Substituting $d = c_0 + \mathrm{Tr}(c_0)$, the expression for λ_0 becomes:

$$\lambda_0 = \sum_{i=0}^{(m-1)/2} (d + c_1 + \lambda_1)^{2^{2i}} = \sum_{i=0}^{(m-1)/2} \left(d + c_1 + \sum_{j=0}^{(m-1)/2} c_1^{2^{2j}} \right)^{2^{2i}}.$$

The expansion of the inner sum allows the interleaving of the consecutive squarings. The analysis can be split in two cases, depending on the format of the extension degree m:

$$\lambda_0 = \begin{cases} c_0 + \displaystyle\sum_{i=0}^{\lfloor m/4 \rfloor - 1} (c_0^{16} + d^4 + c_1^4 + c_1^8)^{2^{4i}} & \text{if } m \equiv 1 \pmod 4 \\ \displaystyle\sum_{i=0}^{\lfloor m/4 \rfloor} (c_0 + d^4 + c_1^2 + c_1^4)^{2^{4i}} & \text{if } m \equiv 3 \pmod 4. \end{cases}$$

The value λ_1 can then be computed as $\lambda_1 = \lambda_0^2 + \lambda_0 + d + c_1$, for a total of approximately m squarings and $m/4$ additions, a cost comparable to a single constant-time half-trace in the base field.

Inversion

Field inversion is implemented by two different approaches based on the Itoh-Tsuji algorithm [IT88]. This algorithm computes $a^{-1} = a^{(2^{m-1}-1)2}$, as proposed in [GP02], with the cost of $m-1$ squarings and a number of multiplications determined by the length of an addition chain for $m-1$. For a variable-time implementation, the squarings for each 2^i-power involved can be converted into multi-squarings [BKNS10], implemented as a trade-off between space consumption and execution time. Each multi-squaring table requires the storage of $2^4 \cdot \lceil \frac{m}{4} \rceil$ field elements. A constant-time implementation must perform consecutive squarings and cannot benefit considerably from a precomputed table of field elements without introducing variance in memory latency, potentially exploitable by an intrusive attacker.

Point addition

The last performance-critical operation to be described is the point addition in λ-affine coordinates. A formula for adding points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ on the curve is proposed in [OLARH14], with associated cost of 2 inversions, 4 multiplications and 2 squarings :

$$x_{P+Q} = \frac{x_P \cdot x_Q(\lambda_P + \lambda_Q)}{(x_P + x_Q)^2}, \qquad \lambda_{P+Q} = \frac{x_Q \cdot (x_{P+Q} + x_P)^2}{x_{P+Q} \cdot x_P} + \lambda_P + 1.$$

Simple substitution of x_{P+Q} in the computation of λ_{P+Q} gives faster new formulas. By unifying the denominators, one field inversion can be traded for 2 multiplications in the formulas below, with associated cost of 1 inversion, 6 multiplications and 2 squarings:

$$x_{P+Q} = \frac{x_P \cdot x_Q(\lambda_P + \lambda_Q)^2}{(x_P + x_Q)^2(\lambda_P + \lambda_Q)}$$
$$\lambda_{P+Q} = \frac{\left[(x_P + x_Q)^2 + x_Q \cdot (\lambda_P + \lambda_Q)\right]^2}{(x_P + x_Q)^2(\lambda_P + \lambda_Q)} + \lambda_P + 1.$$

7.5 Experimental results

The implementation was completed with help of the latest version of the RELIC toolkit [AG]. Random number generation was implemented with the recently introduced `RDRAND` instruction [Cor]. Software was compiled with a prerelease version of GCC 4.9 available in the Arch Linux distribution with flags for loop unrolling, aggressive optimization (`-O3` level) and specific tuning for the Sandy/Ivy Bridge microarchitectures. Table 7.2 presents timings in clock cycles for field arithmetic and Elligator Squared in two different platforms – an Intel Ivy Bridge Core i5 3317U 1.7GHz and a Haswell Core i7 4770K 3.5GHz. The timings were taken as the average of 10^4 executions, with TurboBoost and HyperThreading disabled to reduce randomness in the results.

The constant-time implementation results are mostly for reference: indeed, since the Elligator Squared operation is efficiently invertible, there is no strong reason to compute it in constant time: timing information does not leak secret key data like in the case of a scalar multiplication. However, timing information could conceivably help an active distinguishing attacker; the corresponding attack scenarios are far-fetched, but the paranoid may prefer to choose constant-time arithmetic as a matter of principle.

Table 7.2: Timings for Elligator Squared and underlying field arithmetic in two Intel platforms. Results are in clock cycles and were taken as the average of 10^4 executions with random inputs. FB/VB results refer to generating a random point with fixed-base and variable-base scalar multiplication respectively, using the constant-time, timing-attack protected scalar multiplication from [OLARH14], and computing its Elligator Squared representation with variable-time arithmetic.

Operation	Ivy Bridge	Haswell
Field squaring	13	15
Sparse multiplication	80	44
Multiplication	94	48
Inversion	959	734
Constant-time inversion	1,783	1,610
Half-trace	55	50
Constant-time half-trace	1,213	1,245
Point addition	1,500	1,026
Constant-time point addition	2,367	2,137
Elligator Squared	23,680	22,850
Constant-time Elligator Squared	52,850	51,750
FB with Elligator Squared	127,430	80,180
VB with Elligator Squared	138,480	83,680

7.6 Comparison of Elligator 2 and Elligator Squared on Prime Finite Fields

We have implemented Elligator 2 [BHKL13a] and the corresponding Elligator Squared construction on Curve25519 [Ber06] using the fast arithmetic provided by Bernstein et al. as part of the publicly available implementation of Curve25519 and Ed25519 [BDL+12] in SUPERCOP, in order to compare the two proposed methods on Edwards curves in large characteristic (and to see how they both perform compared to our binary implementation).

To generate a random point and compute the corresponding bitstring representation, the Elligator method requires, on average, 2 scalar multiplications, 2 tests for the existence of preimages and 1 preimage computation. On the other hand, for the same computation, Elligator Squared requires, on average, 1 scalar multiplication, 2 tests for the existence of preimages, 1 preimage computation and 2 computations of the Elligator 2 map function. As a result, compared to the Elligator approach, the Elligator Squared approach requires one scalar multiplication less, but two map function computations more. Therefore, Elligator will be faster than Elligator Squared in contexts where a scalar multiplication is cheaper than two map function evaluations and conversely. Elligator will thus tend to have an edge for protocols using fixed base point scalar multiplication, whereas Elligator Squared will

Table 7.3: Timings for Elligator Squared and Elligator 2 on Curve25519. Results are in clock cycles and were taken as the average of 10^4 executions with random inputs. FB/VB are as in Table 7.2

Operation	Ivy Bridge	Haswell
Scalar multiplication (fixed-base)	42,570	42,180
Scalar multiplication (variable-base, est.)	182,490	162,460
Map function	38,420	36,590
FB with Elligator Squared (on average)	157,500	141,200
FB with Elligator 2 (on average)	114,800	100,200
VB with Elligator Squared (on average, est.)	297,420	261,480
VB with Elligator 2 (on average, est.)	394,640	340,760

perform better for protocols using variable base point scalar multiplication.

This is confirmed by our implementation results, as reported in Table 7.3, which are 35–40% in favor of Elligator in the fixed-base case (FB) but 30–35% in favor of Elligator Squared in the variable-base case (VB). Note that the variable-base scalar multiplication results are estimates based on the SUPERCOP performance numbers on `haswell` and `hydra2`. A comparison with Table 7.2 shows that the binary curve approach is 25% to 200% times faster than the fastest Curve25519 implementation. Observe that our results were obtained using a binary GLS curve with efficient arithmetic implemented in processors with native support to binary field arithmetic and may not translate directly to different parameter choices or computing platforms.

PART III

SECURITY OF PSEUDORANDOM GENERATORS

One of the most fundamental cryptographic primitives is the pseudorandom bit generator. It is a deterministic algorithm algorithm that expands a few truly random bits to a longer sequence of bits that cannot be distinguished from uniformly random bits by a computationally bounded algorithm. It has numerous uses in cryptography, e.g. in signatures schemes or public-key encryption schemes.

As a consequence, pseudorandom generators are one of the main component of security products and their importance for security is hard to overestimate. A number of practical attacks stem from problems with randomness generators. Indeed, it is difficult to obtain randomness on computers and embedded devices, which tend to aggregate various more or less reliable sources of entropy (mouse movements, passwords, network interrupts, electronic noise) and use pseudorandom number generators to expand them into arbitrarily long, hopefully uniform-looking bit strings.

Such practical generators have been analyzed and a framework has been given by Barak and Halevi [BH05] for the Linux generators, who discuss the importance of the randomness collector which maintains a state of enough entropy and an extraction function whose aims is to output random bitstring from the state, which is then expanded into an arbitrarily long pseudorandom string. Cryptographers tend to concentrate on the second aspect: they assume that a pseudorandom generator is seeded with a uniformly distributed bitstring and the goal of the generator is to stretch the seed to a longer bitstring. They define security notions and a generator is called secure if it is hard to distinguish its output from uniformly distributed bitstring given the previous output bits.

The security of the first cryptographically secure generators has been based on some number-theoretic hard problems, in the sense that the problem of distinguishing the outputs from uniform is reduced to a number-theoretic problem. In 1981, Shamir proposed one generator based on the strong RSA problem in [Sha81]; Blum and Micali proposed a generator based on the discrete logarithm problem [BM84]; Blum, Blum and Shub proposed a generator based on the factorization problem in [BBS86]; and Micali and Schnorr defined another generator based on the problem of distinguishing small e-th root modulo a RSA modulus from uniform values in [MS91]. These generators are interesting from a theoretical point of view, but they are rather inefficient and in practice more efficient generators using symmetric cryptographic functions have been preferred.

Number-theoretic pseudorandom generators work by iterating an algebraic map F (public or private) over a residue ring $\mathbb{Z}_\Psi$ on a secret random initial seed value $v_0 \in \mathbb{Z}_\Psi$ to compute values $v_{n+1} = F(v_n) \bmod \Psi$ for $n \in \mathbb{N}$. They output some consecutive bits of the state value v_n at each iteration and their efficiency and security are thus strongly related to the number of output bits. The input v_0 of the generator (and possibly the description of F) is called the *seed* and the output is called the *pseudorandom sequence*.

The last part of this thesis is devoted to contributions in this area and more specifically we analyze the security of some pseudorandom generators. In Chapter 8 we study a scenario where the secret key of any cryptosystem, based on the factorization or the discrete logarithm problem, is generated using a linear congruential generator. We show, in this case, that we can mount an attack to recover the secret key, given the public one and the parameters of the generator, faster than an exhaustive search of the seed. Chapter 9 is dedicated to the cryptanalysis of any nonlinear pseudorandom generator using Coppersmith's techniques. More precisely, we increase the existing bounds leading to the recovery of the seed. Finally we study the security of the Micali-Schnorr generator in Chapter 10 by first presenting some time/memory/data tradeoffs applicable to this particular generator, then by analyzing the statistical properties of its (more or less) algebraic map.

Contents

Chapter 8
Recovering Private Keys Generated With Weak PRNGs

8.1 Introduction

Suppose that the private key of discrete logarithm-based or factoring-based public-key primitive is obtained by concatenating the outputs of a linear congruential generator. How seriously is the scheme weakened as a result?

While linear congruential generators are cryptographically very weak "pseudorandom" number generators, the answer to that question is not immediately obvious, since an adversary in such a setting does not get to examine the outputs of the congruential generator directly, but can only obtain an implicit hint about them—namely the public key.

In this chapter, we take a closer look at that problem, and show that, in most cases, an attack does exist to retrieve the key much faster than with a naive exhaustive search on the seed of the generator.

The problem is similar to the one considered by Bellare, Goldwasser and Micciancio regarding DSA and "pseudorandomness", and this line of work arguably has renewed relevance in view of the sensitive role that random number generation has been found to play in a number of recent noted papers, such as the one by Lenstra et al. at CRYPTO 2012.

This work was presented at IMACC 2013 [FTZ13].

8.1.1 Linear Congruential Generators

Cryptographers have studied the security of the linear congruential generators (LCG), widely used in simulation. These generators are efficient and have a very small memory footprint. Moreover, with suitable parameters, they can have good statistical properties such as a large period length and their output distribution is uniform. As they are also very easy to implement, they tend to be used in the standard libraries of many languages and compilers: the `rand` function proposed in the POSIX standard and the ones used in many C/C++ standard libraries, Java's `java.util.Random`, most implementations of `RAND` in Fortran, etc. Their efficiency and ubiquity make them attractive to implementors, especially in constrained environments, even in some security-sensitive applications.

Unfortunately, LCGs are cryptographically insecure: Boyar [Boy89b] proved that, with a sufficiently long run of the pseudorandom sequence, one can recover the seed in polynomial time in the size of the internal state and Stern [Ste87] proved that this is also the case even if only the most significant bits of each successive states is revealed (see also [Boy89a, JS98, FHK+88]). These attacks are based on lattice reduction, usually LLL [LLL82]. However, Contini and Shparlinski have proposed a careful analysis of these algorithms in [CS05] and established some limits to their applicability, indicating that with properly chosen parameters, the generator might become secure.

8.1.2 Related Work

In any case, these attacks assume that the adversary has direct access to a certain number of outputs (although the parameters of the generator may remain unknown). As a result, using such a cryptographically weak generator in a cryptographic protocol does not automatically make the resulting protocol insecure, because an adversary against the protocol may not have access to the actual outputs of the generator. This led Bellare et al. [BGM97] to analyze the security of the Digital Signature Algorithm (DSA) when the random nonces used in signature generation are computed using a linear congruential generator. They showed that this does seriously break security: a few signatures are enough to recover the secret key. This started an important line of research on the security of DSA when partial information on the nonces is revealed. For instance, Smart and Howgrave-Graham [HGS01] and later Nguyen and Shparlinski [NS02] showed that the knowledge of a small number of the most significant bits of the nonces allows to efficiently recover the secret key using LLL. Bleichenbacher even established [Ble00] that the bias on the single most significant bit of the nonce that occurs when using some version of the NIST generator can be sufficient to efficiently recover the secret key.

8.1.3 Our contributions

In this chapter, we investigate a question similar to the one considered by Bellare et al. but in a different direction than the DSA cryptanalysis papers. We consider the problem of the security of public-key schemes based on the hardness of the discrete logarithm problem or the factoring problem when the secret keys are constructed by concatenating the outputs of a linear congruential generator. Since the attacker does not get those outputs directly, but only an implicit hint, namely the corresponding public key, recovering the secret key is not trivial even if a cryptographically weak generator such as the LCG is used. We show that this is usually enough to recover the secret key much faster than using the trivial exhaustive search on the seed of the generator.

Our attack relies on the assumption that the secret key is obtained as the concatenation of successive outputs of a linear congruential generator. We also assume as is usual in cryptography that the parameters of the LCG are public and therefore an exhaustive search on the seed allows to recover the secret key in time 2^k where k is the size of the seed. Typical parameters for the LCG are 32 bits or 64 bits internal state to allow fast implementation without using a library for large integer arithmetic. The main observation of our work is that if we split the seed of size k into two parts $(A \cdot 2^{k/2} + B)$ where A and B are $(k/2)$-bit long, the linearity of the LCG makes it "almost" possible to write the secret key as a sum $U + V \cdot 2^{k/2}$ where U (respectively V) only depends on B (resp. A) and the parameters of the LCG. This is correct up to carries, which do occur but can be taken care of separately. As a result, we can obtain a time-memory tradeoff on the search for the LCG seed when such generators are used to generate the secret key of a discrete logarithm-based scheme or to the prime factors of an RSA modulus. The discrete logarithm case is mainly a baby-step giant-step attack on the lower and upper halves of the seed, while the factoring case proceeds similarly using multipoint polynomial evaluation.

The main advantage of these attacks is that they allow key recovery from the public key alone, independently of any further information on the underlying cryptographic schemes.

8.1.4 Organization of the chapter

The chapter is organized as follows. After some preliminaries in §8.2, we present our attack in the discrete logarithm case in §8.3 and in the factoring case in §8.4. Finally, in §8.5, we give an overview of the complexity of our attacks for typical parameter sizes.

8.2 Preliminaries

We first recall the definition of the linear congruential generator and fix some notations which will be used in the following sections; then we briefly discussion multipoint evaluation of univariate polynomials, which will be used in the factoring case.

8.2.1 Linear congruential generator

For M an integer of size m bits, we denote by $\mathbb{Z}_M$ the ring of integers modulo M. The internal state of a linear congruential generator evolves according to the following recurrence relation:

$$v_{i+1} = a \cdot v_i + b \bmod M \tag{8.1}$$

where a and b are fixed constant in $\mathbb{Z}_M$ (the parameters of the generator) and $v_0 = s$ is the secret seed. The successive outputs o_i of the generator are the k least (or most) significant bits of v_i at each iteration (for some fixed $k \in \{1, \ldots, m\}$). Note that the recurrence equation is easily solved as:

$$v_i = a^i \cdot s + b \cdot (1 + a + \cdots + a^{i-1}) = a^i \cdot s + b_i \pmod M. \tag{8.2}$$

The following attacks rely on the assumption that a certain secret x is computed as a concatenation of successive outputs of such a linear congruential generator, with known parameters a, b and M. In other words, x can be written as:

$$x = o_0 + 2^k o_1 + \cdots + 2^{(r-1)k} o_{r-1} \tag{8.3}$$

for some fixed constant r. The secret x is then of size rk bits.

8.2.2 Multipoint evaluation of univariate polynomials

Let $P(x) \in \mathbb{Z}_N[x]$, with N an arbitrary integer, be a polynomial of degree less than $d = 2^k$. The multipoint evaluation problem is the task of evaluating P at d distinct points $\alpha_0, \ldots, \alpha_{d-1} \in \mathbb{Z}_N$. Using Horner's rule, it is easy to propose a solution that uses $O(d^2)$ additions and multiplications in $\mathbb{Z}_N$ but it is well-known that one can propose an algorithm with quasi-linear complexity $\widetilde{O}(d)$ operations in $\mathbb{Z}_N$ using a divide-and-conquer approach [Fid72]; a better, more involved algorithm based on the middle product of polynomials has later been proposed in some special cases by Bostan and Schost [BS05, BS04]. That observation has found several applications in cryptanalysis [CN12, CJM+11].

Here is a succinct description of the classical approach, based on product and remainder trees of polynomials. Let $P_0 = \prod_{\ell=0}^{d/2-1}(x - \alpha_\ell)$ and $P_1 = \prod_{\ell=d/2}^{d-1}(x - \alpha_\ell)$ and let us define $R_0 = P \bmod P_0$ and $R_1 = P \bmod P_1$. We have $R_0(\alpha_i) = P(\alpha_i)$ for all $i \in \{0, \ldots, d/2 - 1\}$ and $R_1(\alpha_i) = P(\alpha_i)$ for all $i \in \{d/2, \ldots, d - 1\}$ and this gives immediately a recursive algorithm (i.e. compute P_0, P_1, R_0, R_1 and reduce the problem to the multipoint evaluation of R_0 and R_1 of degree $d/2 = 2^{k-1}$).

Let $A_i(x) = (x - \alpha_i)$ for $i \in \{0, \ldots, d - 1\}$ and $P_{i,j} = A_{j2^i} A_{j2^i+1} \ldots A_{j2^i+2^i-1}$ for $i \in \{0, \ldots, k\}$ and $0 \leq j < 2^{k-i}$. We have $P_{0,j} = A_j$ and $P_{i+1,j} = P_{i,2j} P_{i,2j+1}$ so for

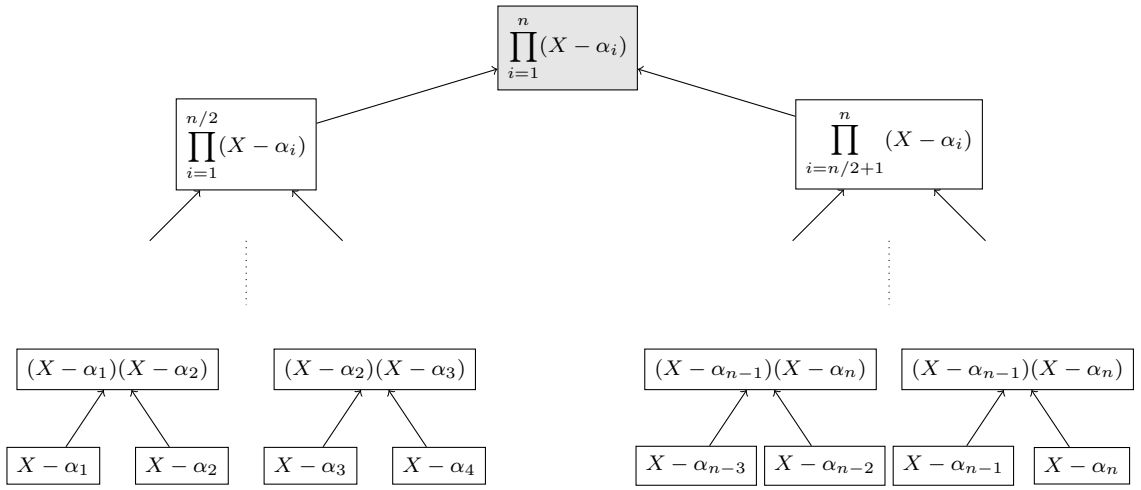

Figure 8.1: Polynomial subproduct tree for the multipoint evaluation algorithm

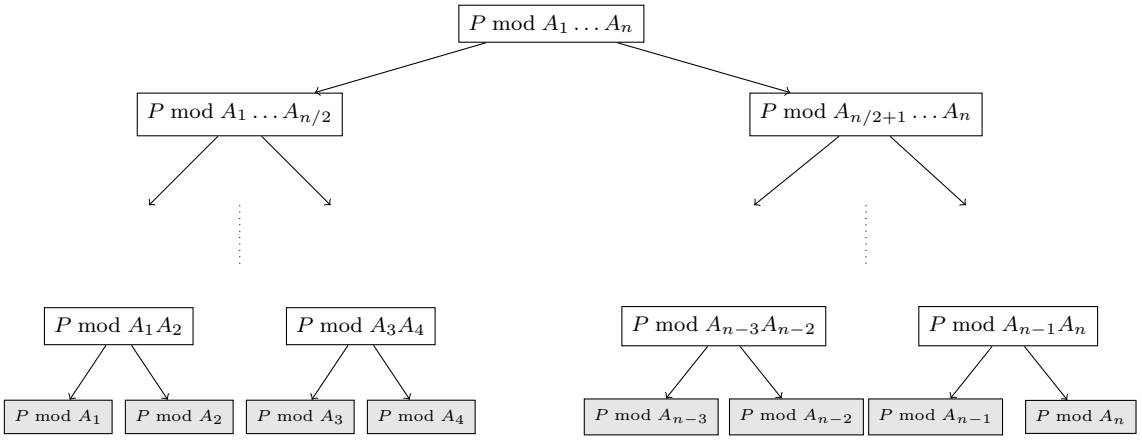

Figure 8.2: Evaluation on the polynomial subproduct tree

$i \in \{0, \dots, k\}$ we can compute recursively all polynomials $P_{i,j}$ and $0 \leq j < 2^{k-i}$ in $2^{k-i-1}O(M(2^i)) = O(M(d))$ operations in $\mathbb{Z}_N$ where $M(i)$ denotes the arithmetic complexity to compute the product of two polynomials of degree i in $\mathbb{Z}_N[x]$. Overall, the computation of all polynomials $P_{i,j}$ requires $O(M(d)\log d)$ operations in $\mathbb{Z}_N$.

The polynomials R_0 and R_1 can be computed using $O(M(d))$ operations in $\mathbb{Z}_N$ (using a Newton inversion), hence the complexity $T(d)$ of the recursive algorithm satisfies $T(d) = 2T(d/2) + O(M(d))$ and therefore $T(d) = O(M(d)\log d)$. Figures 8.1 and 8.2 detail both trees used for this technique.

8.3 The discrete logarithm case

We now consider key generation in a public-key scheme whose security is related to the discrete logarithm problem in some cyclic group $\mathbb{G}$ of prime order q and generator g. Typically, for such a scheme, $\mathbb{G}$, q and g are public parameters, the secret key contains a random element $x \in \mathbb{Z}_q$, and $h = g^x$ is revealed as part of the public key.

Assume that x is obtained from the outputs of a linear congruential generator of known

parameters, as in Equation (8.1). The problem is to recover x from the public data faster than by an exhaustive search on the seed s.

Our approach in a nutshell is as follows. Separate the seed in its lower-order and higher-order halves: $s = u + 2^{k/2} \cdot v$ (we can assume for simplicity's sake that k is even). Then, by Equation (8.1), the internal state of the generator can be written as:

$$v_i = \left(a^i \cdot u + 2^{k/2} \cdot a^i \cdot v + b_i\right) \bmod M.$$

Thus, the corresponding output o_i can essentially be written, in turn, as the sum of a part depending only on u and another part depending only on v—only "essentially" because of possible carry bits and of possible overflows in the addition modulo M, but this can be taken care of, and we will ignore that for the moment.

Then, x is itself of the form $x = U + V$ where U is a publicly computable function of u, and V of v. In the group $\mathbb{G}$, this gives $h = g^U \cdot g^V$, or equivalently:

$$g^U = h \cdot g^{-V}.$$

Now, in time and space $O(2^{k/2})$, we can find a collision between the lists of elements of $\mathbb{G}$ of the form g^U for all $2^{k/2}$ possible values of u on the one hand, and $h \cdot g^{-V}$ for all $2^{k/2}$ possible values of v on the other hand, and hence recover the secret $x = U + V$.

The real algorithm has a slightly higher complexity due to the need to take carries and overflows into account, which we work out below first when $M = 2^k$ (the output is the full internal state) and then in the general case.

Note that since the parameters a and b are known, the constants b_i can be computed publicly and are thus irrelevant to the attack. To simplify notations, we can thus assume without loss of generality that $b = 0$.

Remark. The attack discussed here is generic and can of course be carried out in any cyclic group: it applies to (subgroups of) the multiplicative group of a finite field and to elliptic curves or abelian varieties alike. In the case of an elliptic curve group, the problem is to recover a secret value x from two points P, Q such that $Q = xP$, and when x is obtained from a linear congruential generator as before, it is again possible to divide x into two parts U and V, the first depending on u, the second on v. We can find a collision by checking an equality of the form $Q - UP = VP$.

8.3.1 Attack for non-truncated linear congruential generators

We first work out the details of this attack for a non-truncated linear congruential generator, which satisfies that $M = 2^k$. The non-truncated linear congruential generator is the most efficient of the linear congruential generator in the sense that it outputs the maximal number of available bits at each iteration.

Theorem 8.1. *Given two group elements $g, h \in \mathbb{G}$ with $h = g^x$, where x is an $(r \cdot k)$-bit exponent generated with a non-truncated linear congruential generator with public parameters and k-bit state, there exists an algorithm which retrieves the secret x in time and memory $O(2^{\frac{k+r}{2}})$.*

Proof. As mentioned previously, we may assume without loss of generality that the LCG has parameters such that $b = 0$. By Equation (8.2), its successive outputs are thus of the form:

$$o_i = v_i = (a^i \cdot s) \bmod M = (a^i \cdot s) \bmod 2^k.$$

Now write the seed as $s = u + 2^{k/2} \cdot v$, with u, v of $k/2$ bits. We get:

$$o_i = \left(a^i \cdot (u + 2^{k/2} \cdot v)\right) \bmod 2^k.$$

We can expand that expression for o_i using the following elementary lemma.

Lemma 8.1. *For all $\alpha, \beta, \gamma \in \mathbb{Z}$, $\gamma \neq 0$, there exists $\varepsilon \in \{0, 1\}$ such that:*

$$(\alpha + \beta) \bmod \gamma = (\alpha \bmod \gamma) + (\beta \bmod \gamma) - \varepsilon\gamma.$$

Proof. Indeed, let $L = (\alpha + \beta) \bmod \gamma$ and $R = (\alpha \bmod \gamma) + (\beta \bmod \gamma)$. Clearly, L and R are congruent modulo γ, so they must differ by a multiple of γ. Moreover, $0 \leq L < \gamma$ and $0 \leq R < 2\gamma$, hence $-\gamma < R - L < 2\gamma$, so $R - L$ must be of the form $\varepsilon \cdot \gamma$ with $\varepsilon \in \{0, 1\}$ as required. ∎

Thus, for all indexes i (and any choice of the two seed halves u, v), there exists $\varepsilon_i \in \{0, 1\}$ such that:

$$\begin{aligned} o_i &= (a^i \cdot u \bmod 2^k) + (a^i \cdot 2^{k/2} v \bmod 2^k) - \varepsilon_i \cdot 2^k \\ &= (a^i \cdot u) \bmod 2^k + 2^{k/2}(a^i \cdot v \bmod 2^{k/2}) - \varepsilon_i \cdot 2^k. \end{aligned}$$

If u and v are the two halves of the *correct* seed used to generate x, summing the $2^{ik} o_i$ yields, according to Equation (8.3):

$$x = U + V - Y$$

where:

$$\begin{aligned} U &= \sum_{i=0}^{r-1} 2^{ik} \cdot (a^i u \bmod 2^k) \\ V &= \sum_{i=0}^{r-1} 2^{ik+k/2} \cdot (a^i v \bmod 2^{k/2}) \\ Y &= \sum_{i=0}^{r-1} 2^{(i+1)k} \cdot \varepsilon_i. \end{aligned}$$

We can also decompose Y into a sum $W + Z$ where each of W and Z consist of $r/2$ terms, and obtain the relation $U - Z = x + W - V$, or equivalently:

$$g^{U-Z} = h \cdot g^{W-V}. \tag{8.4}$$

We can thus recover x by finding a collision between two lists of $2^{\frac{k+r}{2}}$ elements of $\mathbb{G}$, namely the g^{U-Z} (for all values of the half-seed u and all possible choices of the bits ε_i in Z) on the one hand, and the $h \cdot g^{W-Z}$ (for all values of the half-seed v and all possible choices of the bits ε_i in W) on the other. Using hash tables, this can be achieved in time and space $O(2^{\frac{k+r}{2}})$.

More precisely, one first computes the $2^{k/2}$ possible values U_i, the $2^{r/2}$ possible values Z_j and stores i, j in a hash table under the key $g^{U_i - Z_j}$. This table contains $2^{\frac{k+r}{2}}$ different values accessible in constant time. Then one computes the $2^{k/2}$ possible values V_s, the $2^{r/2}$ possible values W_t and tests, for each of them, whether $h \cdot g^{W_t - V_s}$ is a key of the hash table. When the test succeeds, one obtains the correct values of u and v and can deduce the value x. The attack is summarized in Algorithm 8.1. ∎

Algorithm 8.1 Attack overview in the discrete logarithm case.

Require: q, g, $h = g^x$, a, b, M
Ensure: x such as $h = g^x$
  Compute the hash table H by storing i, j at $H(g^{U_i - Z_j})$
  for each (V_s, W_t) **do**
    if $H(h \cdot g^{W_t - V_s})$ exists **then return** $x \leftarrow U_i + V_s - Z_j - W_t$
    end if
  end for

8.3.2 Attack for truncated linear congruential generators

We now consider a truncated linear congruential generator with a modulus $M > 2^k$ of size m (with $m < rk$). It is typically less efficient than the non-truncated one, but may have better properties in statistical and security terms. The attack we obtain has a slightly worse complexity than in the non-truncated setting.

Theorem 8.2. *Given two group elements $g, h \in \mathbb{G}$ with $h = g^x$, where x is an $(r \cdot k)$-bit exponent generated with a truncated linear congruential generator outputting the k most (or least) significant bits at each iteration, with public parameters and m-bit state, there exists an algorithm which retrieves the secret x in time and memory $O(2^{m/2} \cdot 5^{r/2})$.*

Proof. The principle of the attack remains similar however the carry is in a larger set of values. As a consequence the complexity in time and in memory is increased. Indeed, starting from Equation (8.2) with $b = 0$, the successive outputs are now of the form:

$$o_i = v_i \bmod 2^k = \big((a^i \cdot s) \bmod M\big) \bmod 2^k$$

in the case where the least significant bits are output, and:

$$o_i = v_i \gg (m-k) = \big((a^i \cdot s) \bmod M\big) \gg (m-k)$$

in the most significant bits case. By writing s as $u + 2^{m/2}v$, we get:

$$v_i = \big(a^i \cdot (u + 2^{m/2} \cdot v)\big) \bmod M.$$

and Lemma 8.1 ensures that:

$$(a^i \cdot u) \bmod M + 2^{m/2}(a^i \cdot v \bmod M) = v_i \text{ or } v_i + M.$$

Using the same lemma and the fact that $o_i = v_i \bmod 2^k$ (LSB case), we obtain:

$$(a^i \cdot u \bmod M) \bmod 2^k + (2^{m/2} \cdot a^i \cdot v \bmod M) \bmod 2^k = \begin{cases} o_i \\ o_i + 2^k \\ o_i + (M \bmod 2^k) \\ o_i + (M \bmod 2^k) + 2^k \\ o_i + (M \bmod 2^k) - 2^k \end{cases}$$

In the MSB case, we have (by denoting $j = m - k$) $o_i = v_i \gg j$ and thus:

$$(a^i \cdot u \bmod M) \gg j + (2^{m/2} \cdot a^i \cdot v \bmod M) \gg j = \begin{cases} o_i \\ o_i - 1 \\ o_i + (M \gg j) \\ o_i + (M \gg j) + 1 \\ o_i + (M \gg j) - 1 \end{cases}$$

Therefore, by applying the attack as before and using Equation (8.4), we have to find a collision between two sets of $2^{m/2} \cdot 5^{r/2}$, the factor $2^{m/2}$ coming from the search of U (respectively V) and the factor $5^{r/2}$ coming from the search of Z (respectively W). ■

8.4 The factoring case

The attacks extend to public-key schemes whose security is related to the hardness of factoring, or of the RSA problem.

Denoting p and q two secret primes obtained from outputs of a linear congruential generator, and N the resulting product published as part of the public key, we would like to find p and q given N and the parameters of the generator.

The idea is again to separate the seed into a lower-order and a higher-order part, and to obtain a time-memory tradeoff compared to exhaustive search. The key ingredient is multipoint polynomial evaluation.

8.4.1 Basic prime generation

We first consider a prime number generation algorithm (see [MOV96]) which consists in, from a random seed, computing the required number of outputs, concatenating them and using a probabilistic primality test such as Miller-Rabin or a deterministic one such as the AKS primality test. If the test fails, one selects another random seed and restarts the algorithm: all primality tests are independent.

As before, we consider the case where the linear congruential generator is not truncated and the case where it is truncated.

Theorem 8.3. *Given a RSA modulus N with $N = pq$, where p is an $(r \cdot k)$-bit prime generated with a non-truncated linear congruential generator (resp. a truncated linear congruential generator outputting the k most or least significant bits at each iteration), with public parameters and k-bit state (resp. m-bit state), there exists an algorithm which factorizes N in time and memory $\widetilde{O}(2^{\frac{k+r}{2}})$ (resp. $\widetilde{O}(2^{m/2} \cdot 5^{r/2})$) with overwhelming probability.*

Proof. For simplicity, we will treat the case of the non-truncated LCG since the use of a truncated one implies only a difference in the exhaustive search of the carry. By splitting

the seed as $s = u + 2^{k/2} \cdot v$, we can write p as $p = U + V - Y$ with:

$$U = \sum_{i=0}^{r-1} 2^{ik}(a^i u \bmod 2^k)$$
$$V = \sum_{i=0}^{r-1} 2^{ik}(2^{k/2}(a^i v \bmod 2^{k/2}))$$
$$Y = \sum_{i=0}^{r-1} 2^{(i+1)k} \cdot \varepsilon_i.$$

We can also cut Y into two $r/2$-bit elements W, Z. Let us denote $A = U - Z$ and $B = V - W$ and suppose having $c(A+B) \bmod N = cp \bmod N$ with c an integer. Then, except the rare case where c is a multiple of q, this value is necessarily a multiple of p. Indeed $cp \bmod pq$ can only have the values 0 (case where $q|c$), $p, \cdots, (q-1)p$. Thus, a greater common divisor computation (GCD) with N will reveal p.

As an attacker, one does not have access to the correct value of the seed. Since u and v can take $2^{k/2}$ distinct values, one can compute the same amount of values U and V. Moreover there are $2^{r/2}$ possibilities for W and Z. In other words, the values $A = U - Z$ and $B = V - W$ are in two sets of $2^{\frac{k+r}{2}}$ elements and we have to find a test in order to determine the good ones.

More precisely, one first computes the $2^{\frac{k+r}{2}}$ different values $B_{s,t}$ by generating the values V_s and W_t and we consider the following polynomial of degree $2^{\frac{k+r}{2}}$:

$$P(X) = \prod_{s,t}(X + B_{s,t}) \bmod N$$

Then one computes the $2^{\frac{k+r}{2}}$ possible values $A_{i,j}$ by generating the values U_i and Z_j and proceeds a multi-evaluation of the polynomial P at the points $A_{i,j}$. The result is a set of $2^{\frac{k+r}{2}}$ values of the form:

$$\Big\{ \prod_{s,t}(A_{i,j} + B_{s,t}) \bmod N \quad | \quad i = 0, \cdots, 2^{k/2} - 1, j = 0, \cdots, 2^{r/2} - 1 \Big\}.$$

Finally one has to compute a test to detect the correct values A and B. It is done by computing a GCD between each value $(A_{i,j})$ and the public modulus N. Since all the values of the seed and all the values of the carry are efficiently tested, the prime p will be recovered except if $P(A)$ is equal to 0. However this failure case is extremely rare: it requires that at least one of the $d-1$ integers composing the product with p is the prime q. The probability is thus equal to $\frac{d-1}{2^{\log q}}$. The attack is summarized in Algorithm 8.2.

■

8.4.2 Improved prime generation: PRIMEINC Method

Since there is no link between each probabilistic primality test in the first prime number generating algorithm, the failed tests are useless and free of cost from the point of view of the attacker. We now propose another one with a link by using a counter.

The PRIMEINC algorithm is a prime number generating algorithm proposed by Brandt and Damgård in [BD92] which basically picks a random number and increases it until a prime is found. In other words, if p is not a prime (but odd), then $p = p + 2$ and repeat. According to the prime number theorem, we expect to find a prime number after $\log p$ trials on average.

Algorithm 8.2 Attack overview in the factorization case.

Require: $N = pq$, a, b, M
Ensure: p such as $N = pq$
  Generate the polynomial $P(X) = \prod_{s,t}(X + V_s - W_t) \bmod N$
  Multi-evaluate P at the points $A_{i,j} = U_i - Z_j$
  for each point $A_{i,j}$ **do**
    if $\gcd(P(A_{i,j}), N) \neq 1$ **then return** $\gcd(P(A_{i,j}), N)$
    end if
  end for

Corollary 8.1. *Considering the two cases of Theorem 8.3 coupled with the PRIMEINC algorithm, there exists an algorithm which factorizes N in time and memory $\widetilde{O}(2^{\frac{k+r}{2}})$ (resp. $\widetilde{O}(2^{m/2} \cdot 5^{r/2})$) with overwhelming probability.*

Proof. In our attack, we now search the correct value of p such as $p = A + B + 2\epsilon$ with $\epsilon \in \{0, \cdots, \log p\}$ (see Remark 8.4.2 for the size of the set). Thus, after the multi-evaluation, our algorithm should have computed a set covering the entire space of search as follows:

$$\Big\{ \prod_{s,t}(A_{i,j} + B_{s,t} + 2\gamma) \bmod N \quad | \quad i \leq 2^{k/2} - 1, j \leq 2^{r/2} - 1, \gamma \leq \log p \Big\}.$$

An efficient way to compute the search of the correct value of γ (i.e. $\gamma = \epsilon$) consists in applying the same trick as before, i.e. writing γ as $\gamma = \gamma_{MSB} + \gamma_{LSB}$ by splitting the bits into two parts.
In other words, in the first part of the algorithm, one computes the different values $B_{s,t}$ and the $\sqrt{\log p}$ values of γ_{LSB}. In the second part, one focus on the different values $A_{i,j}$ and the $\sqrt{\log p}$ values of γ_{MSB}. Thus, after the multi-evaluation, the resulted set corresponds to $2^{\frac{k+r}{2}}\sqrt{\log p}$ values of the form (case $M = 2^k$):

$$\prod_{s,t,\gamma}(A_{i,j} + B_{s,t} + 2(\gamma_{MSB} + \gamma_{LSB})) \bmod N.$$

With overwhelming probability, the value containing p does not contain q too and the test using the greatest common divisor will reveal p.

The modification due to the PRIMEINC method thus increases the complexity in time and in memory by a factor of $\sqrt{\log p}$, which disappears in the $\widetilde{O}$ since $\sqrt{\log p} = O(\sqrt{rk}) = O\big(\frac{r+k}{2}\big)$.

■

Remark. Brandt and Damgård prove in [BD92] that the failure is about equal to $e^{-2\ell}$ when applying $\ell \cdot \log p$ iterations of the PRIMEINC algorithm. In the proof, we put $\ell = 1$, leading to a success rate of 86%.

8.5 Complexity estimates for concrete parameter sizes

Table 8.1 below presents the time complexity of our attack in the discrete logarithm case for typical parameter LCG sizes, as found in implementation of the `random` functions of common compilers and standard libraries, namely a modulus equal to either 2^{32} or 2^{64} (so that modular addition and multiplication can be implemented as simple operations on standard size registers), and an output size equal to either the full modulus size or half of it (corresponding to the top or bottom half of the state). The complexities are to be compared with that of the trivial attack: exhaustive search on seed.

Table 8.1: Overview of our Attacks complexities in the discrete logarithm case. When the output size is smaller than the modulus, the first number corresponds to the LSB case, and the second one to the MSB case.

Secret size	Modulus	Output size	Attack complexity	
160	2^{32}	32	$2^{18.5}$	
160	2^{32}	16	$2^{23.9}$	$2^{27.7}$
160	2^{64}	64	$2^{33.5}$	
160	2^{64}	32	2^{36}	$2^{37.8}$
256	2^{32}	32	2^{20}	
256	2^{32}	16	$2^{28.7}$	$2^{34.6}$
256	2^{64}	64	2^{34}	
256	2^{64}	32	$2^{38.3}$	$2^{41.3}$
512	2^{32}	32	2^{24}	
512	2^{32}	16	$2^{41.4}$	$2^{53.2}$
512	2^{64}	64	2^{36}	
512	2^{64}	32	$2^{44.7}$	$2^{50.6}$
1024	2^{64}	64	2^{40}	
2048	2^{64}	64	2^{48}	

Remarks on results of Table 8.1

Note that the differences of complexity between a linear congruential generator which outputs the least significant bits or the most significant bits is due to the fact that there are only three possibilities for the value of $(a^i \cdot u \bmod M) \bmod 2^k + (2^{m/2} \cdot a^i \cdot v \bmod M) \bmod 2^k$. Indeed, taking $M = 2^{32}$ or $M = 2^{64}$ yields $M \bmod 2^k = 0$.

In a few cases, for 16-bit output size, the complexity is in fact worse than the exhaustive search on the seed. This happens in the truncated case only, when r (the number of LCG outputs used to construct the secret) is particularly large, namely when $5^{r/2}$ (MSB case), resp. $3^{r/2}$ (LSB case), is greater than $2^{m/2}$.

In the factoring case, the complexities are larger by a logarithmic factor, from the use of quasilinear multipoint polynomial evaluation.

Chapter 9
Inferring Sequences Produced by Nonlinear Pseudorandom Number Generators Using Coppersmith's Methods

9.1 Introduction

In this chapter, we revisit the security of number-theoretic generators by proposing better attacks based on Coppersmith's techniques for finding small roots on polynomial equations. Using intricate constructions, we are able to significantly improve the security bounds obtained by Blackburn et al..

This work was presented at PKC 2012 [BVZ12].

9.1.1 Background

Because of the weakness of the linear congruential generator, it was suggested to use a non-linear algebraic map F in order to avoid the attacks but several works [BGPGS03, BGPGS05, BGPGS06, GGI06, GGI05] showed that not too many bits can be output at each stage. Blackburn, Gomez-Perez, Gutierrez and Shparlinski [BGPGS05, BGPGS06] proved that some generators are polynomial time predictable if sufficiently many bits of some consecutive bits of the pseudorandom sequence are revealed (even when F is kept private). It remains open to know what is the maximum quantity of information that can be output for each value v_i allowing the generator to be efficient but still secure against potential attackers

Blackburn et al.'s results are based on a lattice basis reduction attack, using a certain linearization technique. A natural idea – already stated in [BGPGS05] – is instead of using only linear relations in the attack, to use also relations that are derived by taking products of them. This technique was proposed by Coppersmith to find small integer roots of polynomial equations [Cop96a, Cop96b]. In Coppersmith's method, a family of polynomials is first derived from the polynomial whose roots are wanted. This family naturally gives a lattice basis and short vectors of this lattice possibly provide the wanted roots. Blackburn et al. claimed that "this approach does not seem to provide any advantages" and that "it may be very hard to give any precise rigorous or even convincing heuristic analysis of this approach". Our goal in this chapter is to investigate this issue.

9.1.2 Our contributions

We show that if a sufficient number of the most significant bits of several consecutive values v_i of non-linear algebraic pseudorandom generator are given, one can recover the

Table 9.1: Proportions of the most significant bits output from consecutive intermediate state values necessary to predict the generator (comparison between existing bounds and ours)

		Basic proportion		**Asymptotic proportion**	
		Prior result	**Our result**	**Prior result**	**Our result**
Quadratic generator	a,b known	**3/4**	**2/3**	**2/3**	**1/2**
	a,b unknown	**18/19**	**11/12**	**11/12**	**2/3**
Pollard generator	b known	**9/14**	**3/5**	**9/14**	**1/2**
	b unknown	**3/4**	**5/7**	**2/3**	**3/5**

seed v_0 (even in the case where the coefficients of F are unknown). We tackle these issues with Coppermith's lattice-based technique for calculating the small roots of multivariate polynomials modulo an integer. This method is heuristic, which is also the case of some arguments of Blackburn et al. showing that their basic results could be strengthened if the number of pseudorandom bits known to the attacker is increased. If F is a polynomial of degree d known to the attacker, Blackburn et al. [BGPGS06] proved that the generator can be predicted if one outputs a proportion $(d^2-1)/d^2$ of the most significant bits of two consecutive intermediate state values. We improve this result (see section 9.3) by showing that this is also the case if one outputs a proportion as large as $d/(d+1)$ of the most significant bits of two consecutive intermediate state values (or $(d-1)/d$ for sufficiently many consecutive intermediate state values).

In [BGPGS05, BGPGS03], Blackburn et al. focused on the well-known following number-theoretic pseudorandom generators (where p is a prime, $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$):

- The *Quadratic generator* corresponding to the function $F(x) = ax^2 + b \bmod p$
- The *Pollard generator*, a particular case of the quadratic generator when $a = 1$.

Our generic results apply to these settings and improve the previous bounds. Table 9.1 shows a comparison between the results we obtain in this chapter and what is known in the literature. More precisely, it gives the proportion of most significant bits output from each consecutive state values necessary to break the generator in (heuristic) polynomial time. The *basic proportion* corresponds to the case where the adversary knows bits coming from the minimum number of intermediate state leading to a feasible attack; while the *asymptotic proportion* corresponds to the case when the bits known by the adversary come from an infinite number of values.

The bounds on the quadratic generator are described in Section 9.4 and are direct applications of our general results. The bounds on the Pollard generator relies on the *unravelled linearization* technique introduced by Hermann and May in 2009 [HM09] and are described in Section 9.5. An important consequence of our work is that the theoretical data complexity of our attack is decreased compared to the attack from [BGPGS05, BGPGS06, GGI06, GGI05] and if one still wants to use these generators to generate pseudorandom numbers, the efficiency of the schemes is significantly degraded.

9.2 Coppersmith's techniques

In 1996, Coppersmith introduced two lattice-based techniques [Cop96a, Cop96b] for finding small roots on polynomial equations. The first one works for a univariate modular polynomial whereas the second one deals with a bivariate polynomial over the integers. As these techniques had a wide range of cryptanalytic applications, some reformulations and generalizations to more variables have been proposed, see for example Howgrave-Graham's simplifications [HG97, HG01] or also discussions on multivariate tasks [BM05, JM06, BJ07].

All these methods have allowed to attack many instances of public-key cryptosystems. One of the most famous one is due to Boneh-Durfee, breaking the RSA scheme when using a small private key [BD00]. In the following, we give more details explaining how such techniques work in practice. For our purpose, we only focus on the multivariate modular case.

Definition of the problem

Let $f(y_1, \ldots, y_n)$ be an irreducible multivariate polynomial defined over $\mathbb{Z}$, having a root $(x_1, \ldots, x_n)$ modulo a known integer Ψ, that is $f(x_1, \ldots, x_n) \equiv 0 \bmod \Psi$. This root is *small* in the sense that each of its components is bounded by a known value, namely $|x_1| < X_1, \ldots, |x_n| < X_n$. The question thus remains to determine the size on the bounds X_i allowing to recover the desired root in polynomial time.

Collection of polynomials

In a first step, one has to generate a collection of polynomials $f_1, \ldots, f_r$ having $(x_1, \ldots, x_n)$ as a modular root. Usually, we consider multiples and powers of the original polynomial f, namely $f_k = y_1^{k_1} \ldots y_n^{k_n} f^{k_\ell}$, for k in $\{1, \ldots, r\}$. By construction, such polynomials satisfy the relation $f_k(x_1, \ldots, x_n) \equiv 0 \bmod \Psi^{k_\ell}$, i.e. there exists an integer c_k such that $f_k(x_1, \ldots, x_n) = c_k \Psi^{k_\ell}$. From now, let us denote as M the set of monomials appearing in the collection $\{f_1, \ldots, f_r\}$.

Construction of the matrix

The problem of finding small modular roots of f can now be reformulated in a vectorial way. Indeed, each polynomial from our chosen collection can be expressed as a vector over $\mathbb{Z}^t$ by extracting its coefficients and putting them into a vector with respect to a chosen order on M. From now, we construct a matrix $\mathcal{M}$ as described in Figure 9.1 and we define as $\mathbf{L}$ the lattice generated by its rows. On that figure, every row of the upper part is related to one monomial of the set M. The left-hand side contains the bounds on these monomials (*e.g.* the coefficient $X_1^{-1} X_2^{-2}$ is put in the row related to the monomial $y_1 y_2^2$). Each column of the right-hand side contains one of the vectors coming from the initial collection $\{f_1, \ldots, f_r\}$. As this matrix is triangular, the determinant of $\mathbf{L}$ is easily computable:

$$|\det(\mathbf{L})| = \frac{\Psi^{k_1 + \cdots + k_r}}{\prod_{(y_1^{a_1} \ldots y_n^{a_n} \in M)} X_1^{a_1} \ldots X_n^{a_n}}$$

$$\mathcal{M} = \left(\begin{array}{cc|ccc|c} & & f_1 & \dots & f_r & \\ & & \downarrow & \downarrow & \downarrow & \\ 1 & & & & & 1 \\ & X_1^{-1} & & & & y_1 \\ & \ddots & & & & \vdots \\ & X_1^{-a_1} \dots X_n^{-a_n} & & & & y_1^{a_1} \dots y_n^{a_n} \\ \hline & & & & & \\ & 0 & \multicolumn{3}{c|}{} & \\ \cline{4-6} & & \Psi^{k_1} & & & \\ & & & \ddots & & \\ & & & & \Psi^{k_r} & \end{array}\right)$$

Figure 9.1: Initial matrix for Coppersmith's method.

A short vector in a sublattice

Let us now consider the vector $\boldsymbol{r_0} = (1, x_1, \dots, x_1^{a_1} \dots x_n^{a_n}, -c_1, \dots, -c_r)$. By multiplying this vector by the matrix $\mathcal{M}$, one easily obtains:

$$\boldsymbol{s_0} = \left(1, (\frac{x_1}{X_1}), \dots, (\frac{x_1}{X_1})^{a_1} \dots (\frac{x_n}{X_n})^{a_n}, 0, \dots, 0\right)$$

By construction, this vector which, *in some sense*, contains the root we are searching for, belongs to the lattice $\mathbf{L}$. Moreover its norm is very small as $\|\boldsymbol{s_0}\|_2 \leq \sqrt{|M|}$. Thus if we can find such a small vector in the lattice $\mathbf{L}$, then the desired root $(x_1, \dots, x_n)$ will be recovered. In order to do that, we first restrict ourselves to searching in a more appropriated subspace. Indeed, noticing that the last coefficients of s_0 are all equal to zero, we know that this vector belongs to a sublattice $\mathbf{L}'$ whose last coordinates are composed by zero coefficients. By doing elementary operations on the rows of $\mathcal{M}$, one can easily reach that sublattice and prove that its determinant is the same as those of $\mathbf{L}$.

Concluding the method

From that point, one computes an LLL-reduction on the lattice $\mathbf{L}'$ and performs a Gram-Schmidt's orthogonalization on the output basis $(\boldsymbol{b_1}, \dots, \boldsymbol{b_t})$, which gives $(\boldsymbol{b_1^\star}, \dots, \boldsymbol{b_t^\star})$. As $\boldsymbol{s_0}$ belongs to $\mathbf{L}'$, this vector can be expressed as a linear combination of the $\boldsymbol{b_i^\star}$'s. Consequently, if its norm is smaller than those of $\boldsymbol{b_t^\star}$, then $(\boldsymbol{s_0}|\boldsymbol{b_t^\star}) = 0$. Extracting the coefficients appearing in $\boldsymbol{b_t^\star}$, one can construct a polynomial p_1 defined over M such that $p_1(x_1, \dots, x_n) = 0$. By then doing the same process with the vectors $\boldsymbol{b_{t-1}^\star}, \dots, \boldsymbol{b_{t-n+1}^\star}$, this leads to the system $\{p_1(x_1, \dots, x_n) = 0, \dots, p_n(x_1, \dots, x_n) = 0\}$. Under the (heuristic) assumption that all created polynomials are algebraically independent, the previous system can be solved and the desired root recovered in polynomial time.

It thus remains to determine the conditions on the bounds X_i that make this method work. The following lemma is helpful to find these ones.

Lemma 9.1. *Let* $\mathbf{B} = (\boldsymbol{b_1}, \cdots, \boldsymbol{b_n})$ *a LLL-reduced base and* $(\boldsymbol{b_1^\star}, \cdots, \boldsymbol{b_n^\star})$ *the result of the Gram-Schmidt's orthogonalization on* $\mathbf{B}$*. Then:*

$$\|b_n^\star\|_2 \geq 2^{-(n-1)/4}(\det \mathbf{L})^{1/n},$$

where $\mathbf{L} = \langle \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell} \rangle$*.*

The constructed polynomial p_1 will have $(x_1, \ldots, x_n)$ as a root over the integers if the norm of $\boldsymbol{s_0}$ is smaller than those of $\boldsymbol{b_t^\star}$. Knowing that $\boldsymbol{s_0}$ is small and that we have an upper bound on $\|\boldsymbol{b_t^\star}\|_2$ (see Lemma 9.1), the more restrictive condition $\sqrt{|M|} < 2^{-(t-1)/4}(\det(\mathbf{L}))^{1/t}$ will suffice. Removing all parameters that do not influence the asymptotic result, this relation can be simplified to $|\det(\mathbf{L})| > 1$, leading to the following final condition:

$$\prod_{(y_1^{a_1} \ldots y_n^{a_n} \in M)} X_1^{a_1} \ldots X_n^{a_n} < \Psi^{k_1 + \cdots + k_r} \tag{9.1}$$

For such techniques, the most complicated part is the choice of the collection of polynomials, what could be a really intricate task when working with multiple polynomials.

9.3 Attacking a non-linear generator

For Ψ an integer of size π, we denote by $\mathbb{Z}_\Psi$ the field of Ψ elements. A pseudorandom non-linear generator can be defined by the following recurrence sequence:

$$v_{i+1} = F(v_i) \bmod \Psi \tag{9.2}$$

where $F(X) = \sum_{j=0}^{d} c_j X^j$ is a polynomial of degree d in $\mathbb{Z}_\Psi[X]$ and v_0 is the secret seed. We assume that this generator outputs the k most significant bits of v_i at each iteration (with $k \in \{1, \ldots, \pi\}$), i.e. if $v_i = 2^{\pi-k} w_i + x_i$, w_i is output by the generator and $x_i < 2^{\pi-k} = \Psi^\delta$ stays unknown. We want to recover $x_i < \Psi^\delta$ for some $i \in \mathbb{N}$ from consecutive values of the pseudorandom sequence (with δ as large as possible) knowing F or not.

9.3.1 Case F known

Any non-linear pseudorandom generator defined by a known iteration function F can be broken under the condition that sufficiently many bits are output by the generator at each iteration. In the following, we determine that condition when two (theorem 9.1) then more (theorem 9.2) consecutive outputs are known to the attacker.

2 consecutive outputs

Theorem 9.1. *Let* $\mathcal{G}$ *be a non-linear pseudorandom generator defined by a known iteration function* $F(X)$ *of degree* d*. If an adversary has access to two consecutive outputs of* $\mathcal{G}$ *then it will be able to predict the entire sequence that follows ; under the condition that at least* $\frac{d}{d+1}\pi$ *most significant bits are output at each iteration, that is :*

$$\delta < \frac{1}{d+1}$$

Proof. Suppose the adversary is given two approximations w_0 and w_1 of two consecutive values v_0 and v_1 that satisfy (9.2). By denoting v_0 as $2^{\pi-k}w_0 + x_0$ and $v_1 = 2^{\pi-k}w_1 + x_1$, we obtain:

$$2^{\pi-k}w_1 + x_1 - \sum_{j=0}^{d} c_j(2^{\pi-k}w_0 + x_0)^j = 0 \bmod \Psi$$

Let $f(y_0, y_1)$ be the polynomial $y_1 + a_0 + a_1 y_0 + \cdots + a_d y_0^d$ where the values a_i, that explicitly depend on w_0, w_1 and the coefficients c_i, are known to the adversary. The goal is to compute the (small) modular root (x_0, x_1) of $f(y_0, y_1)$ in polynomial time to reach the entire values v_0 and v_1. To do so, let us consider the following collection of polynomials:

$$\{y_0^j f^i(y_0, y_1) \quad | \quad di + j \leq dm \quad \wedge \quad i > 0\}$$

where $m \geq 1$ is a fixed integer. Knowing the shape of f, the list of monomials appearing within this collection can be described as:

$$\{y_1^i y_0^j \quad | \quad di + j \leq dm\}$$

Using Coppersmith's method, the right-hand side (*resp.* the left-hand side) of (9.1) is then equal to:

$$\prod_{i=1}^{m} \prod_{j=0}^{d(m-i)} \Psi^i = \Psi^{\frac{1}{6}m(m+1)(dm-d+3)}$$

$$\left(\textit{resp.} \prod_{i=0}^{m} \prod_{j=0}^{d(m-i)} \Psi^{i\delta}\Psi^{j\delta} = \Psi^{\frac{1}{12}m(m+1)(2d^2m+2dm+6+d^2+d)\delta} \right).$$

Thus, the algorithm (heuristically) outputs the root of f in polynomial time as soon as:

$$\delta < \frac{\frac{1}{6}m(m+1)(dm-d+3)}{\frac{1}{12}m(m+1)(2d^2m+2dm+6+d^2+d)} \xrightarrow[m\to+\infty]{} \frac{1}{d+1} \tag{9.3}$$

what concludes the proof. ■

This bound is better than those previously obtained by Blackburn et al. [BGPGS05]. Indeed, their result was approximately $\delta < 1/d^2$ when two consecutive outputs are known to the attacker.

More consecutive outputs

Theorem 9.2. *Let $\mathcal{G}$ be a non-linear pseudorandom generator defined by a known iteration function $F(X)$ of degree d. If an adversary has access to $n+2$ ($n \geq 1$) consecutive outputs of $\mathcal{G}$ then it will be able to predict the entire sequence that follows ; under the condition that at least $\frac{d^{n+1}-d^n}{d^{n+1}-1}\pi$ most significant bits are output at each iteration, that is :*

$$\delta < \frac{d^n - 1}{d^{n+1} - 1}$$

Proof. Let us assume that the adversary knows $n+2$ consecutive outputs of the generator $w_0, \ldots, w_{n+1}$. Denoting as above $v_i = 2^{\pi-k} w_i + x_i$ (for $i \in \{0, \ldots, n+1\}$) the goal is to compute the (small) solution $(x_0, \ldots, x_{n+1})$ of the multivariate polynomial system:

$$\begin{cases} f_0(y_0, y_1) &= y_1 + a_{00} + a_{01} y_0 + \cdots + a_{0d} y_0^d \bmod \Psi \\ \vdots & \vdots \\ f_n(y_n, y_{n+1}) &= y_{n+1} + a_{n0} + a_{n1} y_n + \cdots + a_{nd} y_n^d \bmod \Psi \end{cases}$$

where each polynomial f_i is constructed in the same way as for the "two consecutive outputs" case. From now one, we use the following collection of polynomials:

$$\left\{ y_0^j f_0^{i_0} \ldots f_n^{i_n} \quad | \quad d(i_0 + di_1 + \cdots + d^n i_n) + j \leq dm \quad \wedge \quad i_0 + \cdots + i_n > 0 \right\}$$

where $m \geq 1$ is a fixed integer. As it seems to be a difficult task to describe the set of monomials appearing in that collection for the general case, we first focus on what happens with two polynomials f_0 and f_1. In that case, the set can be described by the powers of these polynomials, that is

$$\left\{ \underbrace{y_0^j y_1^i}_{f_0} \underbrace{y_1^k y_2^l}_{f_1} \quad | \quad di + j \leq dm \quad \wedge \quad dl + k \leq dm - di - j \right\}$$

Another way of expressing this set is $\left\{ y_0^j y_1^i y_2^l \quad | \quad di + j + dl \leq dm \right\}$. From that point, by induction on n, we can show that the monomials appearing in the collection can be described as:

$$\left\{ y_0^j y_1^{i_0} \ldots y_{n+1}^{i_n} \quad | \quad d(i_0 + di_1 + \cdots + d^n i_n) + j \leq dm \right\}$$

As above, the right-hand side (*resp.* the left-hand side) of (9.1) is then equal to:

$$\prod_{d(i_0+di_1+\cdots+d^n i_n)+j \leq dm} \Psi^{(i_0+\cdots+i_n)} \qquad \left(\textit{resp.} \prod_{d(i_0+di_1+\cdots+d^n i_n)+j \leq dm} \Psi^{\delta(i_0+\cdots+i_n+j)} \right)$$

From now on, it just remains to evaluate the obtained bound on δ. Let us denote $\delta = \frac{A(m,n)}{B(m,n)}$ where:

$$\begin{aligned} A(m,n) &= \sum_{i_0=0}^{m} \sum_{i_1=0}^{\lfloor \frac{m-i_0}{d} \rfloor} \cdots \sum_{j=0}^{d(m-\sum_{p=0}^{n} d^p i_p)} i_0 + \cdots + i_n \\ B(m,n) &= \sum_{i_0=0}^{m} \sum_{i_1=0}^{\lfloor \frac{m-i_0}{d} \rfloor} \cdots \sum_{j=0}^{d(m-\sum_{p=0}^{n} d^p i_p)} i_0 + \cdots + i_n + j \end{aligned}$$

Our goal is to simplify these two expressions, by doing an asymptotic analysis on m. We want to reach a formula only depending on n, the number of outputs. It is quite clear that the floor function appearing in the upper bound of the sums can be omitted as it does not influence the asymptotic result. In that case, the formula becomes:

$$\begin{aligned} A(m,n) &\simeq \sum_{i_0=0}^{m} \sum_{i_1=0}^{\frac{m-i_0}{d}} \cdots \sum_{j=0}^{d(m-\sum_{p=0}^{n} d^p i_p)} i_0 + \cdots + i_n \\ B(m,n) &\simeq \sum_{i_0=0}^{m} \sum_{i_1=0}^{\frac{m-i_0}{d}} \cdots \sum_{j=0}^{d(m-\sum_{p=0}^{n} d^p i_p)} i_0 + \cdots + i_n + j \end{aligned}$$

In order to compute the multiples sums $A(m,n)$ and $B(m,n)$, we will use several times a trick from [HM09] which consists in letting indices of a sum run over a larger range in order to obtain a symmetric formula that is easier to evaluate. Basically, it relies on the following observation which holds for any function f:

$$\sum_{i=0}^{N} f(i) = \frac{1}{d} \sum_{i=0}^{dN} f(\lfloor \frac{i}{d} \rfloor)$$

Indeed, when i runs over $\{1, \ldots, dN\}$, $\lfloor \frac{i}{d} \rfloor$ takes each value in $\{1, \ldots, N\}$ exactly d times. Applying this trick n times on our two expressions and omitting the integer part leads to:

$$\begin{aligned} A(m,n) &\simeq \tfrac{1}{d} \ldots \tfrac{1}{d^n} \sum_{i_0=0}^{m} \sum_{i_1=0}^{m-i_0} \ldots \sum_{j=0}^{d(m-\sum_{p=0}^{n} i_p)} i_0 + \frac{1}{d} i_1 + \cdots + \frac{1}{d^n} i_n \\ B(m,n) &\simeq \tfrac{1}{d} \ldots \tfrac{1}{d^n} \sum_{i_0=0}^{m} \sum_{i_1=0}^{m-i_0} \ldots \sum_{j=0}^{d(m-\sum_{p=0}^{n} i_p)} i_0 + \frac{1}{d} i_1 + \cdots + \frac{1}{d^n} i_n + j \end{aligned}$$

In order to obtain a perfect symmetry of this two formula, we have to remove the factor d in the upper bound of the sum over j. This manipulation depends of the variable we focus on. It's quite easy to understand that for $i_0, \ldots, i_n$, we just have to add a factor d to the front, whereas for j, it's a factor d^2. Thus, we obtain:

$$\begin{aligned} A(m,n) &\simeq d \cdot \tfrac{1}{d} \ldots \tfrac{1}{d^n} \sum_{i_0=0}^{m} \sum_{i_1=0}^{m-i_0} \ldots \sum_{j=0}^{m-\sum_{p=0}^{n} i_p} i_0 + \frac{1}{d} i_1 + \cdots + \frac{1}{d^n} i_n \\ B(m,n) &\simeq d \cdot \tfrac{1}{d} \ldots \tfrac{1}{d^n} \sum_{i_0=0}^{m} \sum_{i_1=0}^{m-i_0} \ldots \sum_{j=0}^{m-\sum_{p=0}^{n} i_p} i_0 + \frac{1}{d} i_1 + \cdots + \frac{1}{d^n} i_n + dj \end{aligned}$$

Let us denote by p_1 the following quantity :

$$p_1 = \sum_{i_0=0}^{m} \sum_{i_1=0}^{m-i_0} \ldots \sum_{j=0}^{m-\sum_{p=0}^{n} i_p} i_0$$

Thanks to the obtained symmetry ours formula, one readily gets:

$$\begin{aligned} A(m,n) &\simeq d \cdot \frac{1}{d} \ldots \frac{1}{d^n} (1 + \frac{1}{d} + \cdots + \frac{1}{d^n}) p_1 \\ B(m,n) &\simeq d \cdot \frac{1}{d} \ldots \frac{1}{d^n} (1 + \frac{1}{d} + \cdots + \frac{1}{d^n} + d) p_1 \end{aligned}$$

We obtain in consequence the following bound:

$$\delta < \frac{d^n - 1}{d^{n+1} - 1}.$$

■

When the number of consecutive values known by the attacker tends to infinity, this condition becomes $\delta < 1/d$. Since d is the degree of the iteration function, this result seems to be the optimal one when using Coppersmith's technique.

9.3.2 Case F unknown

In this section, we show that a non-linear pseudorandom generator defined by an unknown iteration function F can also be break. Theorem 9.3 (respectively the discussion) determines the condition that makes the attack succeed when $d+3$ (respectively more) consecutive outputs are known to the attacker.

$d+3$ consecutive outputs

In order to use Coppersmith's technique to solve that problem, one needs to start the construction with a known polynomial. Knowing that the iteration function F is unknown to the attacker, it has to construct a new polynomial P of degree D (depending on d and on the elimination technique that is used) from the equation $v_{i+1} = F(v_i) \bmod \Psi$. Since there are $d+1$ unknown values (if one assumes that all coefficients appearing in F are non-zero), one requires $d+2$ equations of that form, and thus $d+3$ consecutive outputs of the generator. For instance, one could use the resultant computation technique to proceed to such an elimination.

Theorem 9.3. *Let $\mathcal{G}$ be a non-linear pseudorandom generator defined by an unknown iteration function $F(X)$ of degree d. If an adversary has access to $d+3$ consecutive outputs of $\mathcal{G}$ then it will be able to predict the entire sequence that follows ; under the condition that at least $\frac{D^2(d+3)-1}{D(d+3)}\pi$ most significant bits are output at each iteration, that is $\delta < \frac{1}{D^2(d+3)}$ Moreover, if one assumes that the degree of the leading monomial of P is equal to D, then this bound can be improved to:*

$$\delta < \frac{1}{D(d+3)}.$$

Note that in fact this attack works if there exists a monomial order such that the leading coefficient of F is equal to 1 modulo Ψ. In the general case, this condition is almost always satisfied and this is obviously true when Ψ is prime.

Proof. Let us assume that the attacker knows $w_0, \ldots, w_{d+2}$. By manipulating the system $\left(v_{i+1} = F(v_i) \bmod \Psi, i \in \{0, \ldots, d+1\}\right)$ one obtains an equation $E(v_0, \ldots, v_{d+2}) = 0 \bmod \Psi$. Let us denote P as $E(y_0, \ldots, y_{d+2})$. In that case, the polynomial satisfies $P(x_0, \ldots, x_{d+2}) = 0 \bmod \Psi$. Since the shape of P and its degree D both depend on the technique used to manipulate the initial system, describing the monomials appearing in P and therefore in P^m is an impossible task. Consequently, the only way to perform Coppersmith's method is to choose a simpler but larger set of monomials which necessarily contains those of P^m:

$$\left\{ y_0^{j_0} \ldots y_{d+2}^{j_{d+2}} \quad | \quad j_0 + j_1 + \cdots + j_{d+2} \leq Dm \right\}$$

The leading monomial of P, $LM(P)$, can be described as $y_0^{\alpha_0} \ldots y_{d+2}^{\alpha_{d+2}}$ where at least one of the α_i is non negative. Without loss of generality, we can assume for now that $\alpha_0 > 0$. In that case, one can apply Coppersmith's method on the following collection of polynomials:

$$\left\{ y_1^{j_1} \ldots y_{d+2}^{j_{d+2}} P^i \quad | \quad Di + j_1 + \cdots + j_{d+2} \leq Dm \ \wedge \ 1 \leq i \leq m \right\}$$

As y_0 only comes from the powers of P, the prohibition of the multiplication by y_0 ensures that the collection of polynomials will be linearly independent. The right-hand side (*resp.*

the left-hand side) of (9.1) is then equal to Ψ to the power:

$$\sum_{1\leq i\leq m} \sum_{j_1+\cdots+j_{d+2}\leq Dm-Di} i \qquad \left(resp. \sum_{j_0+\cdots+j_{d+2}\leq Dm} \delta(j_0+\cdots+j_{d+2}) \right).$$

Let us denote $\delta = \frac{A(m,n)}{B(m,n)}$ where:

$$\begin{aligned} A(m,n) &= \sum_{1\leq i\leq m} \sum_{j_1+\cdots+j_{d+2}\leq Dm-Di} i \\ B(m,n) &= \sum_{j_0+\cdots+j_{d+2}\leq Dm} j_0+\cdots+j_{d+2} \end{aligned}$$

As before, our goal consists in simplifying these two expressions by doing an asymptotic analysis on m. In order to obtain perfect symmetric formula, we just have to remove the factor D appearing in the sums on both $A(m,n)$ and $B(m,n)$ expressions. This can be done using the following trick:

$$\begin{aligned} \sum_{j_0+\cdots+j_{d+2}\leq Dm} 1 &\simeq D^{d+3} \sum_{j_0+\cdots+j_{d+2}\leq m} 1 \\ \sum_{j_0+\cdots+j_{d+2}\leq Dm} j_0 &\simeq D^{d+4} \sum_{j_0+\cdots+j_{d+2}\leq m} j_0 \end{aligned}$$

Such simplifications lead to the following formula:

$$\begin{aligned} A(m,n) &\simeq D^{d+2} \sum_{i\leq m} \sum_{j_1+j_{d+2}\leq m-i} i \\ B(m,n) &\simeq D^{d+4} \sum_{j_0+\cdots+j_{d+2}\leq m} j_0+\cdots+j_{d+2} \end{aligned}$$

If we denote by p_1 the following quantity $\sum_{i_1+\cdots+i_{d+3}\leq m} i_1$, then $A(m,n)$ and $B(m,n)$ can be reformulated that way:

$$\begin{aligned} A(m,n) &\simeq D^{d+2}p_1 \\ B(m,n) &\simeq D^{d+4}(d+3)p_1 \end{aligned}$$

In that case, the bound on δ can be expressed as:

$$\delta < \frac{1}{D^2(d+3)}.$$

In fact, this result can be improved if one assumes that the degree of $LM(P)$ is equal to D. Indeed, this monomial can be described as $y_0^{\alpha_0}\dots y_{d+2}^{\alpha_{d+2}}$ with $\sum_{i=0}^{d+2}\alpha_i = D$. In order to keep the linear independency between the polynomials, one should only consider polynomials of the form $Mon \times P^i$ such that $Mon \neq LM(P)$. This leads to the following collection:

$$\left\{ y_0^{j_0}y_1^{j_1}\dots y_{d+2}^{j_{d+2}}P^i \;\middle|\; \begin{array}{l} Di+j_0+j_1+\cdots+j_{d+2}\leq Dm \quad\wedge\quad 1\leq i\leq m \\ \wedge \quad (j_0<\alpha_0)\cup\cdots\cup(j_{d+2}<\alpha_{d+2}) \end{array} \right\}$$

Using the same kind of tricks as before, the resulting asymptotic bound becomes:

$$\delta < \alpha_0\frac{1}{D^2(d+3)} + \cdots + \alpha_{d+2}\frac{1}{D^2(d+3)} = \frac{1}{D(d+3)}.$$

■

More consecutive outputs

We now want to generalize the previous attack when the adversary is given access to more consecutive outputs. Let us assume, for example, that it has access to $d+n+2$ consecutive values $w_0, \ldots, w_{d+1+n}$; its goal is then to compute the (small) solution $(x_0, \ldots, x_{n+d+1})$ of the multivariate polynomial system $(P_1(y_0, \ldots, y_{d+2}), \ldots, P_n(y_{n-1}, \ldots, y_{n+d+1}))$ where the polynomials P_i of degree D, are defined as in the previous section. Exactly as before, finding a general description of the monomials appearing in these polynomials seem to be a challenging task. As a consequence, we consider a larger set of monomials, easier to describe:

$$\left\{ y_0^{j_0} \ldots y_{d+1+n}^{j_{d+1+n}} \quad | \quad j_0 + j_1 + \cdots + j_{d+1+n} \leq Dm \right\}$$

Let us express the leading monomial of P_1 as $y_0^{\alpha_0} \ldots y_{d+2}^{\alpha_{d+2}}$ with at least one of the $\alpha_i \geq 1$, the leading monomial of P_2 as $y_1^{\alpha_0} \ldots y_{d+3}^{\alpha_{d+2}}$ and those of P_n as $y_{n-1}^{\alpha_0} \ldots y_{n+d+1}^{\alpha_{d+2}}$, using a monomial order such as *lex* or *hlex* with $y_0 < \cdots < y_{d+1+n}$. Without loss of generality, we can assume that $\alpha_0 > 0$. From that, one can apply Coppersmith's method on the following collection of polynomials:

$$\left\{ y_n^{j_1} \ldots y_{n+d+1}^{j_{d+2}} P_1^{i_1} \ldots P_n^{i_n} \;\middle|\; \begin{array}{l} D(i_1 + \cdots + i_n) + j_1 + \cdots + j_{d+2} \leq Dm \\ \wedge \quad 1 \leq i_1 + \cdots + i_n \leq m \end{array} \right\}$$

The prohibition of the multiplication by $y_0, \ldots, y_{n-1}$ ensures that all the polynomials of the collection are linearly independent. Thus, the right-hand side (*resp.* the left-hand side) of (9.1) is equal to Ψ to the power:

$$\sum_{1 \leq i_1 + \cdots + i_n \leq m} \; \sum_{j_1 + \cdots + j_{d+2} \leq Dm - D(i_1 + \cdots + i_n)} i_1 + \cdots + i_n$$
$$\left(\textit{resp.} \sum_{j_0 + \cdots + j_{n+d+1} \leq Dm} \delta(j_0 + \cdots + j_{n+d+1}) \right).$$

By using the same kind of tricks as in the proof of Theorem 9.3, one can show that the resulting asymptotic bound is:

$$\delta < \frac{n}{D^{n+1}(n+d+2)}.$$

Remark. Of course, this bound is not interesting as its value decreases when the adversary is given access to more outputs. But in fact we are convinced that it can significantly be improved. Indeed, using the same kind of techniques as in the previous case, we will probably be able to gain a factor D for each involved polynomial, thus reaching the bound:

$$\delta < \frac{n}{D(n+d+2)} \xrightarrow[m \to +\infty]{} \delta < \frac{1}{D}$$

In practice we notice that this conjecture seems to be true, see for instance the analysis of the quadratic generator in section 9.4.1.

9.4 Application: Attacking the quadratic generator

For p a prime of size π, we remind that the notation $\mathbb{Z}_p$ refers to the field of p elements. The quadratic generator is defined by the following recurrence sequence:

$$v_i = av_{i-1}^2 + b \bmod p \tag{9.4}$$

In that particular case, the iteration function $F(x)$ is defined as $F(x) = ax^2 + b$ where $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$ are constant values. Exactly as before, we denote the secret seed as $v_0 \in \mathbb{Z}_p$ and we assume that the generator outputs the k most significant bits of v_i at each iteration (with $k \in \{1, \ldots, \pi\}$). In other words, each value v_i can be written as $2^{\pi-k}w_i + x_i$ where w_i is output by the generator and $x_i < 2^{\pi-k} = p^\delta$ stays unknown. Our goal consists in recovering the value $x_i < p^\delta$ for some $i \in \mathbb{N}$ by using some consecutive values output by the pseudorandom sequence (with δ as large as possible).

9.4.1 Case F known

If the adversary is given access to two consecutive outputs of the generator, then it can break the scheme under the condition that sufficiently many bits are output by the generator at each iteration. More precisely, for a fixed value m (that will define the size of the corresponding lattice), the bound on δ should respect the following condition, directly coming from equation (9.3) in theorem 9.1:

$$\delta < \frac{1}{6} \cdot \frac{2m+1}{m+1}$$

In particular, taking $m = 1$ leads to the bound $\delta < 1/4$ previously reached by Blackburn et al. [BGPGS05]. This bound can be improved to $\delta < 1/3$ when the quantity m goes to infinity. This value is exactly the same as those previously obtained by Blackburn et al. when the authors assume that the adversary is given access to an infinite number of outputs, whereas it only requires here two outputs of the generator. Finally, when increasing the number of known outputs to infinity, the condition becomes $\delta < 1/2$ (see theorem 9.2).

9.4.2 Case F unknown

4 consecutive outputs

Because the coefficients a and b appearing in the iteration function $F(x) = ax^2 + b$ are unknown to the attacker, the first step consists in expressing the relations between the outputs of the generator exclusively in terms of known quantities. More precisely, by using four consecutive outputs, the adversary is able to eliminate the quantities a and b by considering the following polynomial P of degree 3:

$$\begin{aligned} P =& c + c_0y_0 + c_1y_1 + c_2y_2 + c_3y_3 + d_0(y_0^2 - y_1^2) + d_1(y_2y_0 - y_0y_3) + d_2(y_1^2 - 3y_2^2 + 2y_1y_2) \\ &+ d_3(y_2^2 - 3y_1^2 + 2y_1y_3) + e(y_2^2y_1 - y_1^3 + y_1^2y_3 - y_2^3 - y_0^2y_3 + y_0^2y_2) \bmod p \end{aligned}$$

Since each coefficient in this polynomial is inversible modulo p, one can consider that $LM(P)) = 1$. Thus, applying theorem 9.3, one reaches the bound $\delta < 1/15$, knowing that the degree d of the iteration function F is equal to 2 and those of the polynomial P is 3. In fact, this bound can be improved as the coefficient related to x in the iteration function, is equal to zero. Indeed, the denominator in the formula given by theorem 9.3 can be

expressed as $D.\ell(n)$ where $\ell(n)$ is the number of required outputs. In this particular case, as $\ell(n)$ is equal to 4, the bound thus becomes $\delta < 1/12$. In the same scenario, Blackburn et al. [BGPGS05] reached the value $\delta < 1/19$.

More consecutive outputs

Let us now assume that the adversary is given access to more consecutive outputs. We generalize the previous construction using the fact that the iteration function F contains one zero coefficient. In that case, if the set of monomials stays easy to formulate, this is not the case for the collection of polynomials. Indeed, when the set of monomials is given by $\left\{ y_0^{j_0} \dots y_{n+2}^{j_{n+2}} \quad | \quad j_0 + j_1 + \dots + j_{n+2} \leq 3m \right\}$, the collection is:

$$\left\{ y_0^{j_0} y_1^{j_1} y_2^{a_2} \dots y_{n+1}^{a_{n+1}} y_{n+2}^{j_{n+2}} P_1^{i_1} \dots P_n^{i_n} \;\middle|\; \begin{array}{l} 0 < i_1 + \dots + i_n \leq m \\ \wedge \quad 0 \leq a_\ell \leq \min(2, 3m - 3\sum_{t=1}^{n} i_t - \sum_{t=2}^{\ell-1} a_t) \\ \text{for } \ell \in \{2, \dots, n+1\} \\ \wedge \quad j_0 + j_1 + j_{n+2} \leq K \quad \text{with} \\ K = 3m - 3(i_1 + \dots + i_n) - (a_2 + \dots + a_{n+1}) \end{array} \right\}$$

The estimation of the "weight" of these two sets is a challenging task, thus we consider a smaller but simpler collection by restricting the condition on $i_1 + \dots + i_n$:

$$\left\{ y_0^{j_0} y_1^{j_1} y_2^{a_2} \dots y_{n+1}^{a_{n+1}} y_{n+2}^{j_{n+2}} P_1^{i_1} \dots P_n^{i_n} \;\middle|\; \begin{array}{l} 0 < i_1 + \dots + i_n \leq \lfloor m - \frac{2}{3}n \rfloor \\ \wedge \quad 0 \leq a_\ell \leq 2 \quad \text{for } \ell \in \{2, \dots, n+1\} \\ \wedge \quad j_0 + j_1 + j_{n+2} \leq K \quad \text{with} \\ K = 3m - 3(i_1 + \dots + i_n) - (a_2 + \dots + a_{n+1}) \end{array} \right\}$$

This set is contained in the previous one but we are convinced that it gives the same asymptotical bound even if the initial collection probably gives better results for small values of n and m. Let us now determine the corresponding bound δ. To make the proof more clear, we denote δ as $\frac{A(m,n)}{B(m,n)}$ where:

$$A(m,n) = \sum_{1 \leq i_1 + \dots + i_n \leq \lfloor m - 2/3n \rfloor} \; \sum_{a_2, \dots, a_{n+1} \leq 2} \; \sum_{j_0 + j_1 + j_{n+2} \leq 3m - \sum_{\ell=2}^{n+1} a_\ell - 3\sum_{k=1}^{n} i_k} i_1 + \dots + i_n$$

$$B(m,n) = \sum_{j_0 + \dots + j_{n+2} \leq 3m} j_0 + \dots + j_{n+2}$$

As before, our goal consists in simplifying these two expressions by doing an asymptotic analysis on m. First of all, the sum over i_j in the quantity $A(m,n)$ can be taken from 1 to m instead of $\lfloor m - 2/3n \rfloor$. Indeed, the value $2/3n$ will not influence the highest power of m appearing in this expression, it can then be omitted for the asymptotic analysis. By using the same kind of argument, one can easily remove the value $\sum_{\ell=2}^{n+1} a_\ell$ in the upper bound of the sum over j_0, j_1 and j_{n+2}. In that case, $A(m,n)$ can be rewritten as:

$$A(m,n) = \sum_{1 \leq i_1 + \dots + i_n \leq m} \; \sum_{0 \leq a_2, \dots, a_{n+1} \leq 2} \; \sum_{j_0 + j_1 + j_{n+2} \leq 3m - 3\sum_{k=1}^{n} i_k} i_1 + \dots + i_n$$

Table 9.2: Asymptotic bounds for the quadratic generator when F is unknown.

Number of outputs	4	5	6	7	8	9	10	11	12
Asymptotic bound	1/12	2/15	1/6	4/21	5/24	2/9	7/30	8/33	1/4

The next step will consist in removing the sums over the coefficients $a_2, \ldots, a_{n+1}$ from the expression corresponding to $A(m,n)$, what is permitted as they do not appear anywhere in the expression. Making such a simplification allows us to rewrite $A(m,n)$ that way:

$$A(m,n) = 3^n \sum_{1 \leq i_1 + \cdots + i_n \leq m} \; \sum_{j_0 + j_1 + j_{n+2} \leq 3(m - \sum_{k=1}^{n} i_k)} i_1 + \cdots + i_n$$

In this formula, one can see the appearance of a factor 3^n due to the fact that the coefficients a_i were taken their values from the set $\{0,1,2\}$, thus reaching three possibilities. From now on, it just remains to remove the factor "3" appearing in the sum over j on both $A(m,n)$ and $B(m,n)$ expressions. This can be done using the following trick:

$$\begin{aligned} \sum_{j_0 + j_1 + j_{n+2} \leq 3m} 1 &\simeq 3^3 \sum_{j_0 + j_1 + j_{n+2} \leq m} 1 \\ \sum_{j_0 + j_1 + j_{n+2} \leq 3m} j_0 &\simeq 3^4 \sum_{j_0 + j_1 + j_{n+2} \leq m} j_0 \end{aligned}$$

Such simplifications lead to the following formula:

$$\begin{aligned} A(m,n) &= 3^{n+3} \sum_{i_1 + \cdots + i_n \leq m} \; \sum_{j_0 + j_1 + j_{n+2} \leq m - \sum_{k=1}^{n} i_k} i_1 + \cdots + i_n \\ B(m,n) &= 3^{n+4} \sum_{j_0 + \cdots + j_{n+2} \leq m} j_0 + \cdots + j_{n+2} \end{aligned}$$

If we denote by p_1 the following quantity $\sum_{i_1 + \cdots + i_{n+3} \leq m} i_1$, then $A(m,n)$ and $B(m,n)$ can be reformulated that way:

$$\begin{aligned} A(m,n) &= 3^{n+3} n p_1 \\ B(m,n) &= 3^{n+4}(n+3) p_1 \end{aligned}$$

In that case, the bound on δ can be expressed as:

$$\delta < \frac{3^{n+3} \cdot n p_1}{3^{n+4}(n+3) p_1}$$

When n tends to infinity, we obtain $\delta < \frac{1}{3}$. This value seems to confirm the conjecture $\delta < 1/D$ discussed in remark 9.3.2. Moreover, it significantly improves the bound $\delta < 1/12$ previously obtained by Blackburn et al. in [BGPGS05]. Additionally, this method provides interesting asymptotic bounds for small values of n (when m goes to infinity), see Table 9.2.

$$\mathcal{M} = \left(\begin{array}{ccccc|cc|c} 1/U & & & & & 1 & 0 & u \\ & 1/X & & & & 1 & 0 & x \\ & & 1/X^2 & & & 0 & 1 & x^2 \\ & & & 1/UX & & 0 & 2 & ux \\ & & & & 1/U^2 & 0 & 1 & u^2 \\ \hline & & & & & \Psi & & \\ & & & 0 & & & \Psi^2 & \end{array}\right)$$

Figure 9.2: First trick of the *unravelled linearization* technique.

9.5 The Pollard generator

The recursive sequence of the Pollard generator is defined as $v_i = v_{i-1}^2 + b \bmod p$ with $b \in \mathbb{Z}_p$. In other words, it is a particular instance of the quadratic generator where the constant a is equal to 1. As a consequence, the attack scenario is exactly the same as in the previous section when b is known to the attacker. However, if one takes advantage of the fact that a is fixed to 1, a specific analysis can be made and a better bound can be obtained. To reach such a result, we use a novel technique, called unravelled linearization whose description is provided below.

9.5.1 Unravelled linearization

In 2009, Hermann and May [HM09] introduced a new technique called *unravelled linearization* that allows to work with smaller lattices by optimizing the way the initial polynomial is written. The idea consists in improving the bounds, see equation (9.1), by reducing the number of monomials in the set M while keeping the same amount of powers of Ψ appearing in the right hand side of the equation.

Let us show what happens on a toy example, say $f(x, y) = x^2 + x + y$ having a root (x_0, y_0) modulo Ψ where $|x_0| < X$ and $|y_0| < Y$ with $X = Y$. The idea is to find the better way of linearizing f before proceeding to Coppersmith's construction. Let us fix, as an example, $u = x^2$. In that case, the polynomial f becomes $g(u, x, y) = u + x + y$ and the bounds on the root can de determined by the following formula $UXY < \Psi$. Knowing that $U = X^2$, this leads to $X = \Psi^{1/4}$. Now, let us take another smarter linearization, say $u = x^2 + y$, leading to the polynomial $g(u, x) = u + x$. This time, the formula becomes $UX < \Psi$, what leads to the improved bound $X = \Psi^{1/3}$. This result can easily be understood by the fact that the "weight" of y is hidden in u by the weight of x^2.

There is another tricky manipulation to do in order to conclude. Let us consider again our toy example $g(u, x) = u + x$ and let us construct the original matrix defined by Coppersmith taking the collection $\{g, g^2\}$. This leads to the matrix $\mathcal{M}$ (see Figure 9.2), thus reaching the asymptotic bound $U^4X^4 < \Psi^3$, what gives $X < \Psi^{1/4}$.

But here is the point: by definition of u, the monomial x^2 can easily be written as $u - y$, thus allowing to express the polynomial g^2 as $g^2 = u^2 + 2ux + u - y$. Such a manipulation leads to the matrix $\mathcal{M}'$ (see Figure 9.3). In this case, the obtained bounds on the root

$$\mathcal{M}' = \left(\begin{array}{ccccc|cc|c} 1/U & & & & & 1 & 1 & u \\ & 1/X & & & & 1 & 0 & x \\ & & \mathbf{1/Y} & & & 0 & -1 & \mathbf{y} \\ & & & 1/UX & & 0 & 2 & ux \\ & & & & 1/U^2 & 0 & 1 & u^2 \\ \hline & & & & & \Psi & & \\ & & 0 & & & & \Psi^2 & \end{array}\right)$$

Figure 9.3: Second trick of the *unravelled linearization* technique.

can be reformulated as $U^4X^2Y < \Psi^3$ what gives the improved result $X < \Psi^{3/11}$. This benefit can be understood by the fact that we have managed to decrease the weight of the monomials in the set M by 1 while keeping the exact number of powers of Ψ appearing in the right hand side of equation (9.1). Such manipulations are quite hard to proceed, they strongly rely on the linearization chosen for the initial polynomial f.

9.5.2 Case F known

2 consecutive outputs

Let us first assume that the adversary is given access to two consecutive outputs of the generator, namely w_0 and w_1. Knowing that $v_0 = 2^{\pi-k}w_0 + x_0$ and $v_1 = 2^{\pi-k}w_1 + x_1$, we reach the same relation as those previously obtained for the quadratic case:

$$x_1 - 2^{\pi-k+1}w_0x_0 - x_0^2 - b + 2^{\pi-k}w_1 - 4^{\pi-k}w_0^2 = 0 \bmod p$$

Let us denote by $f(y_0, y_1)$ the polynomial $y_1 - c_0y_0 - y_0^2 + d_0$ where the coefficients $c_0 = 2^{\pi-k+1}w_0$ and $d_0 = -b + 2^{\pi-k}w_1 - 4^{\pi-k}w_0^2$ are known to the attacker. As usual, its goal consists in recovering the small modular root (x_0, x_1) of $f(y_0, y_1)$.

To solve this problem, we use the unravelled linearization technique. As already stated, the first step consists in choosing a good linearization for f. In this particular case, we set $u = y_1 - y_0^2$, what leads to the following polynomial $g(y_0, u) = u - c_0y_0 + c \bmod p$. In that case, the bound on u can thus be expressed as $U = X_0^2$.

Let us now consider the collection of polynomials defined as $y_0^j g^i(y_0, u)$ with $i + j \leq m$ and $i > 0$. The list of monomials appearing in that collection can be described as $M = \left\{ y_0^j u^i \quad | \quad i + j \leq m \right\}$. Initially, we use this set of polynomials to construct the matrix defined by Coppersmith, as in section 9.2. In that case, the right-hand side (*resp.* the left-hand side of) of (9.1) can easily be expressed as p to the power

$$\sum_{i=1}^{m}\sum_{j=0}^{m-i} i = \frac{1}{6}m^3 + o(m^3) \qquad \left(\textit{resp. } \delta \sum_{i=0}^{m}\sum_{j=0}^{m-i} 2i + j = \frac{\delta}{2}m^3 + o(m^3) \right)$$

The idea of the unravelled linearization technique is to improve the bound on δ by decreasing the weight of the monomials. To do so, one should proceed to a "back-substitution"

in the constructed matrix, as explained in the previous section. In that particular case, knowing that $y_0^2 = y_1 - u$, the following replacement is done (for all monomials μ such that $\mu \cdot y_0^2 \in M$): $\mu \cdot y_0^2 \rightarrow \mu \cdot y_1 - \mu \cdot u$. It is obvious that the presence of $\mu \cdot y_0^2$ in the set M implies those of $\mu \cdot u$. As a consequence, doing such a manipulation allows to replace the quantity $\mu \cdot y_0^2$ by $\mu \cdot y_1$ thus decreasing by "1" the weight on the monomials. If we express the collection M as $M = \left\{ y_0^{2b+a} u^i \quad | \quad a \in \{0,1\} \wedge a + 2b + i \leq m \right\}$, after the back-substitution, we obtain the set $M' = \left\{ y_1^b y_0^a u^i \quad | \quad a \in \{0,1\} \wedge a + 2b + i \leq m \right\}$. In that case, the *new* left-hand side in equation (9.1) becomes p raised to the power:

$$\delta \sum_{a=0}^{1} \sum_{i=0}^{m-a} \sum_{b=0}^{\lfloor \frac{m-i-a}{2} \rfloor} (a + b + 2i) = \delta \frac{5}{12} m^3 + o(m^3)$$

Thus, the corresponding asymptotic bound on δ becomes:

$$\delta < \frac{1/6m^3 + o(m^3)}{5/12m^3 + o(m^3)} \xrightarrow[m \to +\infty]{} \frac{2}{5}.$$

This bound is better than those previously obtained by Blackburn et al. [GGI06]. Indeed, in this paper the authors managed to reach the asymptotic bound $\delta < 5/14$ when working with one polynomial. One can also notice that the fraction $2/5$ is exactly the bound obtained in [HM09] for attacking the Blum-Blum Shub generator using the same kind of trick.

More consecutive outputs

When working with more outputs, one can easily generalize the method explained before in the same way as what has been done for the Blum-Blum Shub generator, thus reaching the bound $\delta < 1/2$.

9.5.3 Case F unknown

3 consecutive outputs

Let us now consider the case of an adversary having access to three consecutive outputs of the generator. In that case, writing two consecutive recurrence relations and substracting both of them leads to: $-x_1^2 + x_0^2 + x_2 + c_0 x_0 - c_1 x_1 + c = 0 \bmod p$ with:

$$\begin{aligned} c_0 &= 2^{\pi - k + 1} w_0 \\ c_1 &= (2^{\pi - k + 1} w_1 + 1) \\ c &= 2^{\pi - k} w_2 - 2^{\pi - k} w_1 + 4^{\pi - k} w_0^2 - 4^{\pi - k} w_1^2 \end{aligned}$$

The goal is to recover the small modular root (x_0, x_1, x_2) of the polynomial $f(y_0, y_1, y_2) = -y_1^2 + y_0^2 + y_2 + c_0 y_0 - c_1 y_1 + c$. To do so, we use again the unravelled linearization technique. In order to linearize the polynomial f, we set $u = -y_1^2 + y_0^2 + y_2$ to reach the new following expression $g(u, y_0, y_1) = u + c_0 y_0 - c_1 y_1 + c$. Let us now consider the collection of polynomials defined as $y_0^k y_1^j g^i$ with $i + j + k \leq m$ and $i > 0$. In that case, the list of involved monomials can easily be expressed as $M = \left\{ u^i y_1^j y_0^k \quad | \quad i + j + k \leq m \right\}$. Thus, the right-hand side of Coppersmith's equation (9.1) is given by p to the power:

$$\sum_{i=1}^{m} \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} i = \frac{1}{24} m^4 + o(m^4).$$

Instead of just evaluating the weight of the monomials appearing in M, we want to perform some back-substitutions before. In this case, the rule given by the linearization is (for all monomials μ such that $\mu \cdot y_1^2 \in M$): $\mu \cdot y_1^2 \rightarrow \mu \cdot y_0^2 + \mu \cdot y_2 - \mu \cdot u$. One can easily notice that the presence of the monomial $\mu \cdot y_1^2$ in the set M automatically implies those of $\mu \cdot y_0^2$ and $\mu \cdot u$. As a consequence, the previous linearization rule allows to replace a monomial of the form $\mu \cdot y_1^2$ by one of those $\mu \cdot y_2$ in the constructed matrix, again decreasing by "1" the weight on the involved monomials. The shape of the *new* constructed set of monomials is then:

$$\left\{ u^i y_2^b y_1^a y_0^k \quad | \quad a \in \{0,1\} \wedge i + k + a + 2b \leq m \right\}$$

In that case, the *new* left hand side of equation (9.1) becomes:

$$\delta \sum_{a=0}^{1} \sum_{i=0}^{m-a} \sum_{b=0}^{\lfloor \frac{m-i-a}{2} \rfloor} \sum_{k=0}^{m-i-a-2b} (a + b + 2i + k) = \frac{7\delta}{48} m^4 + o(m^4)$$

which leads to the following bound on δ:

$$\delta < (1/24m^4 + o(m^4))/(7/48m^4 + o(m^4)) \xrightarrow[m \to +\infty]{} 2/7.$$

More consecutive outputs

Let us now assume that the attacker knows $n+2$ consecutive outputs of the Pollard generator for $n \geq 2$. As before, we denote by f_i the relation between two outputs defined as

$$f_i = 2^{\pi - k} w_i + y_i - (2^{\pi - k} w_{i-1} + y_{i-1})^2 - b \bmod p \quad i \in \{1, \ldots, n\}$$

These polynomials have (x_i, x_{i-1}) as a root modulo p and if we denote by $g_i = f_{i+1} - f_i$ for $i \in \{1, \ldots, n\}$, we have $g_i = -y_i^2 + y_{i-1}^2 + y_{i+1} + c_i y_{i-1} - d_i y_i + e_i \bmod p$ for some constants c_i, d_i, e_i known to the adversary.

Each of these polynomials involves three consecutive outputs of the generator. Knowing the set of polynomials $\{g_1, \ldots, g_n\}$, the attacker wants to recover the unknown values x_i. To do so, we will use again the unravelled linearization technique. In that particular case, we decide to choose $u_i = -y_i^2 + y_{i-1}^2 + y_{i+1}$, thus leading to the following equations: $g_i = u_i + c_i y_{i-1} - d_i y_i + e_i$. Consider now the following collection of polynomials

$$\left\{ y_1^k y_0^j g_1^{i_1} \ldots g_n^{i_n} \quad | \quad k + j + \sum_{l=1}^{n} 2^{l-1} i_l \leq m \wedge i_1 + \ldots i_n > 0 \right\}$$

In that case, the right hand side of equation (9.1) can be expressed as p raised to the power:

$$\sum_{i_1=0}^{m} \cdots \sum_{k=0}^{m - \sum_{p=1}^{n} 2^{p-1} i_p - j} i_1 + \cdots + i_n$$

The set of monomials appearing in that collection is equal to (using the same trick as in section 9.3.1):

$$M = \left\{ y_1^{i_1} y_0^{j_1} u_1^{k_1} y_2^{j_2} u_2^{k_2} \ldots y_n^{j_n} u_n^{k_n} \quad | \quad i_1 + \sum_{l=1}^{n} 2^{l-1} (j_l + k_l) \leq m \right\}$$

Now, let us use the following back-substitution rule (for $\mu \cdot y_i^2 \in M$): $\mu \cdot y_i^2 \to -\mu \cdot u_i + \mu \cdot y_{i-1}^2 + \mu \cdot y_{i+1}$. As before, when $\mu \cdot y_i^2$ appears in the collection, it is easy to see that it is also true for the monomials $\mu \cdot y_{i-1}^2$ and $\mu \cdot u_i$. As a consequence, such an operation allows to replace each element of the form $\mu \cdot y_i^2$ by $\mu \cdot y_{i+1}$ thus decreasing by 1 the weight on the monomials in M. Note that this will not necessarily be the case if one considers any rule for the back-substitution step, see Section 9.6 for a more detailed discussion on that point. The final step thus consists in determining the new set we obtain after doing that back-substitution. In fact, the collection of polynomials has been selected such that every monomial of the form $\mu \cdot y_{i+1}$ already belongs to the set M, the only exception being for elements involving the last variable, say $\mu \cdot y_{n+1}$.

To understand that, let us come back to the case of two polynomials. In this scenario, the set of monomials is defined as $\{y_1^{i_1} y_0^{j_1} u_1^{k_1} y_2^{j_2} u_2^{k_2} \quad | \quad i_1 + j_1 + k_1 + 2j_2 + 2k_2 \leq m\}$. If $i_1 = 2b_1 + a_1$, then the use of the back-substitution changes the description into $y_1^{a_1} y_1^{2b_1} y_0^{j_1} u_1^{k_1} y_2^{j_2} u_2^{k_2} \rightsquigarrow y_1^{a_1} y_2^{b_1} y_0^{j_1} u_1^{k_1} y_2^{j_2} u_2^{k_2}$. Using the same argument as in section 9.3.1, this new set can be rewritten as:

$$\left\{ y_1^{a_1} y_0^{j_1} u_1^{k_1} y_2^{j_2} u_2^{k_2} \quad | \quad a_1 = 0, 1 \wedge a_1 + j_1 + k_1 + 2j_2 + 2k_2 \leq m \right\}$$

The shape of this set shows that each monomial appearing after the back-substitution step $\mu \cdot y_1^2 \to \mu \cdot y_2$ was already present in the initial collection. From now, it just remains to consider the case $\mu \cdot y_2^2 \to \mu \cdot y_3$. By fixing $j_2 = a_2 + 2b_2$ as before, we obtain the following new collection:

$$\left\{ y_1^{a_1} y_2^{a_2} y_0^{j_1} u_1^{k_1} u_2^{k_2} y_3^{b_2} \quad | \quad a_1, a_2 \in \{0, 1\} \wedge a_1 + 2a_2 + j_1 + k_1 + 2k_2 + 4b_2 \leq m \right\}$$

This concludes the analysis for the case of two polynomials g_1, g_2. By recurrence, the set of monomials appearing after all back-substitutions for n polynomials can be expressed as:

$$\left\{ y_1^{a_1} \dots y_n^{a_n} y_0^{j_1} u_1^{k_1} \dots u_n^{k_n} y_{n+1}^{b_n} \quad | \quad a_1, \dots, a_n \in \{0, 1\} \wedge j_1 + \sum_{\ell=1}^{n} 2^{\ell-1}(a_\ell + k_\ell) + 2^n b_n \leq m \right\}$$

The left hand side of Coppersmith's equation (9.1) is thus p to the power:

$$\delta \sum_{0 \leq a_1, \dots, a_n \leq 1} \sum_{j_1 + \sum_{\ell=1}^{n} 2^{\ell-1}(a_\ell + k_\ell) + 2^n b_n \leq m} (\sum_{\ell=1}^{n} (a_\ell + 2k_\ell) + j_1 + b_n)$$

Coppersmith's method allows to compute the solution of our multivariate system under the condition:

$$\delta < \frac{\sum_{i_1=0}^{m} \dots \sum_{k=0}^{m - \sum_{l=1}^{n} 2^{l-1} i_l - j} i_1 + \dots + i_n}{\sum_{0 \leq a_1, \dots, a_n \leq 1} \sum_{j_1 + \sum_{l=1}^{n} 2^{l-1}(a_l + k_l) + 2^n b_n \leq m} a_1 + \dots + a_n + j_1 + 2(k_1 + \dots k_n) + b_n}$$

In order to make the proof more clear, we write the bound δ as $\frac{A(m,n)}{B(m,n)}$ where:

$$A(m,n) = \sum_{i_1=0}^{m} \dots \sum_{i_n=0}^{\lfloor \frac{m - \sum_{p=1}^{n-1} 2^{p-1} i_p}{2^{n-1}} \rfloor} \sum_{j=0}^{m - \sum_{p=1}^{n} 2^{p-1} i_p} \sum_{k=0}^{m - \sum_{p=1}^{n} 2^{p-1} i_p - j} i_1 + \dots + i_n$$

$$B(m,n) = \sum_{0 \leq a_1, \dots a_n \leq 1} \sum_{j_1=0}^{m - \sum_{p=1}^{n} 2^{p-1} a_p} \sum_{k_1=0}^{m - j_1} \dots \sum_{b_n=0}^{\lfloor \frac{m - j_1 - \sum_{p=1}^{n} 2^{p-1}(a_p + k_p)}{2^n} \rfloor} \mathbf{b}$$

with $\mathbf{b} = a_1 + \cdots + a_n + j_1 + 2(k_1 + \cdots + k_n) + b_n$. We will simplify these two expressions by doing an asymptotic analysis on m. First of all, for $B(m,n)$, we notice that the quantity $a_1 + \cdots + a_n$ doesn't affect the highest power of m and can thus be removed. The same argument allows us to omit the coefficients $2^{p-1}a_p$ in the upper bound of the sums over $j_1, \ldots, b_n$, on the one hand, and to remove the floor function, on the other hand. Finally each sum over a_i can be replaced by a factor 2, what gives a global factor 2^n. At this step, doing all these changes gives:

$$\begin{aligned} A(m,n) &= \sum_{i_1=0}^{m} \cdots \sum_{i_n=0}^{\frac{m-\sum_{p=1}^{n-1} 2^{p-1} i_p}{2^{n-1}}} \sum_{j=0}^{m-\sum_{p=1}^{n} 2^{p-1} i_p} \sum_{k=0}^{m-\sum_{p=1}^{n} 2^{p-1} i_p - j} i_1 + \cdots + i_n \\ B(m,n) &= 2^n \sum_{j_1=0}^{m} \sum_{k_1=0}^{m-j_1} \cdots \sum_{k_n=0}^{\frac{m-j_1-\sum_{p=1}^{n-1} 2^{p-1} k_p}{2^{n-1}}} \sum_{b_n=0}^{\frac{m-j_1-\sum_{p=1}^{n} 2^{p-1} k_p}{2^n}} j_1 + 2(k_1 + \cdots + k_n) + b_n \end{aligned}$$

We now have to do the same trick as in the first proof in order to obtain a symmetric formula easier to evaluate. Here we reach:

$$\begin{aligned} A(m,n) &= \frac{1}{2} \cdots \frac{1}{2^{n-1}} \sum_{i_1=0}^{m} \cdots \sum_{i_n=0}^{m-\sum_{p=1}^{n-1} i_p} \sum_{j=0}^{m-\sum_{p=1}^{n} i_p} \sum_{k=0}^{m-\sum_{p=1}^{n} i_p - j} i_1 + \frac{1}{2} i_2 + \cdots + \frac{1}{2^{n-1}} i_n \\ B(m,n) &= 2^n \cdot \frac{1}{2} \cdots \frac{1}{2^n} \sum_{j_1=0}^{m} \sum_{k_1=0}^{m-j_1} \cdots \sum_{k_n=0}^{m-j_1-\sum_{p=1}^{n-1} k_p} \sum_{b_n=0}^{m-j_1-\sum_{p=1}^{n} k_p} \mathbf{b}' \end{aligned}$$

with $\mathbf{b}' = j_1 + 2(k_1 + \frac{1}{2}k_2 + \cdots + \frac{1}{2^{n-1}}k_n) + \frac{1}{2^n}b_n$. Let us denote by p_1 the following quantity:

$$p_1 = \sum_{i_1=0}^{m} \sum_{i_2=0}^{m-i_1} \cdots \sum_{i_{n+2}=0}^{m-\sum_{p=1}^{n+1} i_p - j} i_1$$

In these expressions, each variable plays the same role as the others, thus our two expressions can be rewrote as follows:

$$\begin{aligned} A(m,n) &= \tfrac{1}{2} \cdots \tfrac{1}{2^{n-1}} (1 + \cdots + \tfrac{1}{2^{n-1}}) \cdot p_1 \\ B(m,n) &= \tfrac{1}{2} \cdots \tfrac{1}{2^{n-1}} (1 + 2(1 + \cdots + \tfrac{1}{2^{n-1}}) + \tfrac{1}{2^n}) \cdot p_1 \end{aligned}$$

As a consequence, we obtain the following bound:

$$\delta < \frac{\frac{2^n - 1}{2^{n-1}}}{1 + \frac{2^{n+1}-2}{2^{n-1}} + \frac{1}{2^n}}$$

When $n \to \infty$, we find $\delta < \frac{2}{5}$.

In that particular case, we think this bound could be improved to $\delta < 1/2$, following the discussion from remark 9.3.2.

9.6 Discussion on the unravelled linearization technique

In Hermann and May's paper [HM09], the unravelled linearization technique is described as follows: one first has to find a good way of linearizing the initial polynomial, then

Table 9.3: Difference between two back-substitution for the Pollard's attack with two polynomials.

Initial values for $p = 512$ bits		$u_1 \to y_2 - y_1^2 + y_0^2$ $u_2 \to y_3 - y_2^2 + y_1^2$		$y_1^2 \to y_2 - u_1 + y_0^2$ $y_2^2 \to y_3 - u_2 + y_1^2$	
Value of m	Size of sublattice	Theo. output	Exp. output	Theo. output	Exp. output
4	46	357	358	357	358
5	80	354	356	354	355
6	130	352	353	351	352
7	200	350	351	349	350

the back-substitution step should be proceed on the monomials of the set M. What has not been said is that there exists many ways of doing the second step, say the back-substitution, depending on the monomials that appear in the linearization. To illustrate that fact, let us reconsider, as an example, what has been done for the Pollard's attack when b is unknown and having three outputs. In that case, we remind that we had the following linearization: $u = y_0^2 - y_1^2 + y_2$. From that formula, there are two ways of doing the back-substitution, namely $u \to y_0^2 - y_1^2 + y_2$ or $y_1^2 \to y_0^2 - u + y_2$. If these two rules seem to be exactly the same, in fact, they do not work the same way. Indeed, in the first case, the presence of u in the set of monomials do not necessarily imply those of y_0^2 (resp. y_1^2), what leads to particular conditions, which is not the case for the second rule. Despite that, with only one linearization this difference do not influence the result. But when the number of polynomials grows, that is the number of linearization rules, the choice of the back-substitution will play a role for intermediate bounds and even asymptotic ones. Table 9.6 presents the difference that can occur when two different rules are applied for the back-substitution step, in the particular case of two polynomials.

Chapter 10
Security of the Micali-Schnorr generator

10.1 Introduction

In this chapter we first study two cryptographic computational problems related to the RSA problem, using time/memory/data tradeoffs. Given a modulus N, product of two large prime numbers, and an odd exponent e, coprime to $\varphi(N)$ the order of the multiplicative group $\mathbb{Z}_N^*$, the RSA problem consists in recovering the plaintext $x \in \mathbb{Z}_N^*$ from a random ciphertext $y = x^e \bmod N$. The variants we look at consider particular instances of this problem where the plaintext is small or where the plaintext is small and only a part of the ciphertext is known. These two problems appear to be related to the security of a pseudorandom generator proposed by Micali and Schnorr.

In a second time, we study the statistical properties of the k least (or most) significant bits of $x^e \bmod N$, where N is an RSA modulus and x only belongs to a small interval of $[0, N)$. We then provide the first rigorous evidence that the cryptographic pseudo-random generator proposed by Micali and Schnorr is based on firm foundations. This proof is missing in the original paper and does not cover the parameters chosen by the authors. Consequently, we extend the proof to get a new result closer to these parameters using recently new exponential sums results and we show some limitations of our technique.

This work was presented at COCOON 2013 [FVZ13] and COCOON 2014 [FZ14].

10.1.1 The Micali-Schnorr generator

Micali and Schnorr [MS91] proposed a variant of the RSA generator that on a secret random initial seed value $x_0 \in \mathbb{Z}_N^*$ computes the intermediate values $v_i = x_i^e \bmod N$ and outputs, for some $k \in \mathbb{N}$, the k least significant bits of v_i. But the successor x_{i+1} of x_i is formed from a separate part of v_i, the remaining most significant bits (contrary to the *incestuous* RSA generator where $x_{i+1} = v_i$). The security of Micali-Schnorr pseudorandom generator relies on the (strong) assumption that the distribution of $x^e \bmod N$ for random k-bit integers is indistinguishable from the uniform distribution on $\mathbb{Z}_N^*$. The generator is insecure if $(1 - 1/e) \log N$ least significant bits are output per iteration but no better attack was proposed since its proposal 25 years ago. It remains open to know what is the maximum quantity of information that can be output per iteration allowing the generator to be efficient but still secure against potential attackers.

10.1.2 The RSA assumption

The RSA assumption states that, given a random value y in $\mathbb{Z}/N\mathbb{Z}$ where N is the product of two large primes, it is difficult to compute a e-th root of y, i.e. find x such that $y = x^e \bmod N$. The RSA problem has been heavily studied by mathematicians and no attack

more efficient than factoring the RSA modulus has been found since its discovery. Usually, it is very difficult to prove a computational assumption such as RSA and cryptographers try to prove that this one is at least as difficult as another one, for instance factoring. However, some evidences for the non-equivalence of these two hard problems has been provided by Boneh et Venkatesan in [BV98] while the RSA assumption seems to hold.

RSA is a valid cryptographic assumption since on average its difficulty seems to be established thanks to its self-reducibility property. Indeed, it is well-known that if we are able to invert RSA on a non-negligible subset of $\mathbb{Z}/N\mathbb{Z}$, then we can invert nearly all values in $\mathbb{Z}/N\mathbb{Z}$ with high probability. Based on this assumption, cryptographers have proposed and proved that the RSA signature and encryption schemes using adequate padding functions [BR96, BR94] are secure in the random oracle model [BR93]. The security proof of RSA-OAEP for encryption appeared in [FOPS01].

Another direction to assess the security of a computational assumption consists in showing that the values we are looking for are computationally or statistically indistinguishable from the uniform distribution on bitstrings of the same size. Consequently, the best the adversary can do is to guess this value until he finds it. For RSA, it is easy to see that the value y is uniformly distributed if x is. Here we are interested in the short RSA problem: given y and the promise that $x < M \ll N$, find x such that $x^e \bmod N$.

10.1.3 Time/memory tradeoffs

As dynamic programming, time/memory tradeoffs is a well-known technique to reduce the time complexity of a problem using memory. Shamir and Schroeppel in [SS81] have described such algorithms for specific NP-complete problems such as knapsack problems. In cryptography, this technique has been used many times to analyze the security of symmetric primitives such as block ciphers or stream ciphers and some computational problems such as the baby-step giant-step algorithm to compute discrete logarithms. Basically, some computations can be done independently of other resources. For instance, using the public key the adversary can precompute some values and store a small fraction of these values in the offline phase. Then, the adversary gets some ciphertexts and his goal can be to recover the secret key.

In [Hel80] Hellman described a technique to invert random looking functions. This technique has been rigorously studied in [FN99] by Fiat and Naor to work for any functions and rigorous lower bounds have been given in [BBS06] by Barkan, Biham and Shamir. Oeschlin in [Oec03] described a variant of Hellman tradeoff, but this variant has been show equivalent to Hellman tradeoff by Barkan et al. since many heuristics can be applied to Hellman technique. Finally, Babbage [Bab95] and Golic [Gol97], then Biryukov and Shamir [BS00] presented tradeoff for stream cipher by using more or less data. This resource is a crucial parameter in cryptanalysis and it is important to present attacks using as low data complexity as possible.

10.1.4 Our contributions

In the first part of this chapter, we use time/memory/data tradeoff techniques to propose algorithms for two computational problems related to the security of the Micali-Schnorr pseudorandom generator. The algorithms are decomposed into two phases: the preprocessing one where the attacker constructs large hash tables using the structure of the focused cryptosystem, and the realtime phase where it uses the data produced by the cryptosystem and the hash tables to retrieve the secrets. The three tradeoffs algorithms we describe are similar to the tradeoffs for stream ciphers. However, in order to construct such algorithms,

we need to specify the function f we used. For stream ciphers, the main idea is to execute from a hidden state the generator in order to have at least $\log S$ bits of output if the state is of bitsize S. Here, we decide to truncate the output value. It is a bit weird to define f in such a way since the iteration of such functions is no more related to the iteration of the generator. However, the only things we need is to cover the space in such a way that the inversion will be possible. This choice of function f is suitable for Micali-Schnorr generator but does not work for the Blum-Blum-Shub or the RSA generator. Moreover, in order to prove that the many Hellman tables algorithm works (our second algorithm), we need to prove that each table uses an independent function. We provide such claim in the analysis of the second algorithm. Indeed, this independence assumption is in fact the tricky part of the analysis and Hellman paper relies on heuristic in order to provide lower bounds on the time complexity of his scheme. Using a computational argument we prove that the considered functions are independent.

Our algorithms do not contradict the strong assumption used for the Micali-Schnorr pseudorandom generator. They can be applied even though only a small part of the generator is output at each iteration. Moreover, we will show that once one value is recovered using the algorithms we describe for the first problem, then we are also able to retrieve the seed by using another time/memory tradeoff.

In the second part of this chapter, we will prove the following informal theorem for different values of M.

Theorem 10.1. *Let $N = pq$ be a balanced RSA modulus, e the public exponent and $M < N$. Let the function $f : \mathbb{Z}/M\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$ defined as $f(x) = x^e \mod N$. The k least significant bits of $f(x)$ for $k < \log N$ are statistically indistinguishable from the uniform distribution on $\{0,1\}^k$.*

For $M \gg \sqrt{N}$, we will show it using classical bounds, and for $N^{1/e} \ll M \ll \sqrt{N}$, we will use more recent results proved by Wooley [Woo12]. This last bound is very close to be optimal since for $M \leq N^{1/e}$, it is possible in polynomial-time to recover $x \leq M$ given the $k \geq M$ least significant bits of $f(x)$. Indeed, in this case the function f becomes non modular and the problem of retrieving x is quite easy by using Hensel's lifting.

We then show an application of these theorems for the Micali-Schnorr generator. Micali and Schnorr original proof refers to the first bound $M \gg \sqrt{N}$ when $e = 3$, or the second one otherwise since they proposed $M = N^{2/e}$. However the proof is missing and they do not give any hint to explain their more aggressive choice of parameters. Indeed, it would be possible to output less bits at each iteration of the generator, but the efficiency of this generator would be less efficient than the Blum-Blum-Shub generator [BBS86] for instance. Micali and Schnorr prefer to output more bits and avoid the previous attack when $M = N^{1/e}$. Our result allows us to propose parameters ensuring the randomness of the output. We also explain that the parameters proposed by Micali and Schnorr are close to be optimal in the special case of $e = 3$.

10.1.5 Outline

In Section 10.2, we present the first problem we look at and basics about the Micali-Schnorr pseudorandom generator. We explain why the problem is easy for some small parameters. In Section 10.3, we describe two time/memory algorithms for solving the first problem using different tradeoffs. In Section 10.4, we show other tradeoffs to recover the seed of the generator. Finally, in Section 10.5 we study Theorem 10.1 and propose an application for the Micali-Schnorr generator.

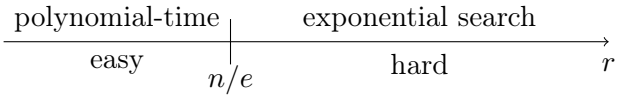

Figure 10.1: Difficulty of the problem depending of the value r

10.2 Micali-Schnorr Pseudorandom Generator

Let (e, N) a RSA public key with e small compared to $\log N$ and $x_0 \in [0, 2^r)$ with $2^r \ll N$ a secret seed of bitsize r. The Micali-Schnorr pseudorandom generator proposed in [MS91] is defined as follows:

$$v_i = x_{i-1}^e \mod N \quad \text{and} \quad v_i = 2^k x_i + w_i \quad \text{for} \quad i \geq 1.$$

At each iteration, this generator outputs the k least significant bits of v_i, denoted by w_i. In addition, denoting n the bitsize of the modulus N, only x_i of bitsize $r = n - k$, unknown, is reused for the next iteration. Since the generator outputs $O(k/\log e)$ bits per multiplication, one wants k to be as large as possible and e to be as small as possible to be efficient. This pseudorandom generator is proven secure under the following strong assumption:

Assumption 10.1. *The distribution of $x^e \bmod N$ for random r-bit integers x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of elements of $(\mathbb{Z}/N\mathbb{Z})^*$.*

Clearly this assumption cannot be true if one does not restrain the tests to polynomial-time ones only because of the lack of entropy in input. Micali and Schnorr have proposed the parameters $r = 2n/e$ and thus $k = n(1 - 2/e)$ which are very aggressive parameters in order to increase the efficiency of the generator.

Description of the problem. Let (e, N) the RSA public key with N of bitsize n. Using the equality $v_i = 2^k x_i + w_i$ where $v_i \in \mathbb{Z}_N$, $w_i \in [0, 2^k)$ and $x_i \in [0, 2^r)$, we consider the recurrence sequence

$$\forall i \geq 1, \quad v_i = x_{i-1}^e \mod N \tag{10.1}$$

Given (e, N, r), $\{w_1, \cdots, w_j\}$ with $j \in \mathbb{N}$, the problem consists in retrieving one value x_c with $c \in \{0, \cdots, j-1\}$.

For an attacker, finding one of the values x_i using some iterations of the Micali-Schnorr pseudorandom generator will lead to infer its next outputs. The difficulty of the above problem depends highly on the value of r. Figure 10.1 sums up this hardness, with a transition value equal to n/e. We first explain why it is easy to solve the problem when the bitsize of r is less than n/e.

Theorem 10.2. *Suppose that the value x_0 of bitsize r is odd. If $r \leq n/e$, given (e, N) and w_1, there exists a polynomial-time algorithm which retrieves the value x_0.*

Proof. If $r \leq n/e$, the modular reduction is not performed in Equation (10.1), so $v_1 = x_0^e$ over the integers and using $v_1 = 2^k x_1 + w_1$, one has the following modular equation:

$$x_0^e = w_1 \mod 2^k$$

where all the values except x_0 are known. We now use the well-known Hensel's lifting lemma to retrieve this secret value.

Lemma 10.1 (Hensel's lifting lemma)**.** *Let p be a prime and c be a positive integer. One denotes f a polynomial having a root x modulo p^c which satisfies:*

$$f(x) = 0 \mod p^c \quad \text{and} \quad f'(x) \neq 0 \mod p$$

Then, one can lift x to obtain an unique nontrivial root $x^* \in [0, p^{c+1})$ verifying:

$$f(x^*) = 0 \mod p^{c+1} \quad \text{and} \quad x^* = x \mod p^c$$

With Lemma 10.1, by using $f(x) = x^e - w_1$, one can reconstruct bit per bit x_0 looking at the powers of 2. The value x^* can be efficiently computed by $x^* = x + \lambda \cdot 2^c$ where $\lambda = -\frac{f(x)}{2^c} \cdot (f'(x))^{-1} \bmod 2$.

■

Note that if the value x_0 is even, one loses the uniqueness of the lift. However, computing $x^e - w_1 \mod 2^k$ for each candidate x of bitsize r can suffice to retrieve this value; one can also test another output w_i of the generator. Another possibility to retrieve the seed consists in raising w_1 to the power $e^{-1} \bmod 2^{k-1}$ (notice that e is odd). However the complexity of Hensel lifting is linear in the size of the root, contrary to this exponentiation. To avoid this simple algorithm but to remain efficient, i.e. to output a maximum of bits per iteration, the parameter k has to be smaller than $\lfloor n(1 - \frac{1}{e}) \rfloor$. Finally, it seems hard to find a polynomial-time algorithm if $r > n/e$, for example by using Coppersmith techniques, which are techniques bases on lattice reduction to find small modular roots.

10.3 Solving the Problem using Time/Memory/Data Tradeoffs

For now, we consider the problem in the case where r is larger than n/e and we will present two similar algorithms that use different tradeoffs in order to solve the problem. These algorithms use the fact that only the hidden information, i.e the value x_i of a relatively small bitsize r, is recycled for the next iteration contrary to some other pseudorandom generators as the BBS or the RSA ones. We denote the five key parameters as follows:

- 2^r represents the cardinality of the search space.
- P represents the time required by the preprocessing phase of the algorithm.
- M represents the quantity of access memory required for the algorithm.
- T represents the time required by the online phase of the algorithm.
- D represents the quantity of data required for the algorithm.

10.3.1 First algorithm

The first algorithm is quite simple to explain and to implement but not really efficient. The preprocessing phase consists in storing the couples $(x, \text{LSB}_k(x^e \mod N))$ for some different values of x in a hash table. During the online phase, one tests for each value w_i if it appears in the hash table. For example, it will work by taking $M = 2^{r/3}$ and $D = T = 2^{2r/3}$ or even $M = T = D = 2^{r/2}$.

Theorem 10.3. *Given (e, N) and D consecutive values $w_1, \cdots, w_D$, there exists an algorithm which retrieves one of the values $x_0, \cdots, x_{D-1}$ in time T, by using M random access memory such that $TM = O(2^r)$ with $D = O(T)$.*

Proof. We first detail the algorithm and then its complexity.
Algorithm. The preprocessing phase of the first algorithm is quite simple: one takes p random different values $x_0^1, \cdots, x_0^p$ and computes $w_1^i = LSB_k((x_0^i)^e \mod N)$ for $i = 1, \ldots, p$ where $LSB_k(x)$ represents the k least significant bits of x. The storage of these values can be done using a hash table where the keys are the w_1^i and their associated values the x_0^i.
During the online phase of the algorithm, one tests for each value $w_i \in \{w_1, \cdots, w_D\}$ if it has an image for the hash function. If the test succeeds, this image represents the unknown value of the precedent iteration ; the problem is then solved, i.e the pseudorandom generator becomes from this step fully predictable. If the test fails, one tries with an other output.
Complexity. The preprocessing time P is equal to $O(p)$. For each value w_i, the probability that a collision occurs with a value in the hash table is equal to $p/2^r$. Then, one can expect a collision when the number of tested values D is such that $Dp = 2^r$. Moreover, the amount of random access memory M and the time required by the online phase of the algorithm T are easily quantifiable since $M = p \cdot O(1)$ and $T = D \cdot O(1)$, leading to the tradeoff $TM = O(2^r)$ with $D = O(T)$.

■

Remark. During the preprocessing phase, even if P is larger than $2^{r/2}$, the birthday paradox is not a problem because the values are stocked: one can impose the uniqueness of each of them. However, during the online phase, if $D > 2^{r/2}$, one has to manage with this paradox. The problem consists in having $D > 2^{r/2}$ distinct values over 2^r possibilities, which is the coupon collector's problem [Tuc97]. A mathematical analysis shows that one needs $O(D \cdot log(D))$ values to achieve the algorithm.

10.3.2 Second algorithm using Hellman's tables

The next algorithm is based on [Hel80, BS00]. Hellman then Biryukov and Shamir have proposed different attacks using tradeoffs for breaking block ciphers and stream ciphers. We define a special function in order to apply these attacks for solving our problem, and thus for the Micali-Schnorr pseudorandom generator. With this second algorithm, one can consider for instance the parameters $P = T = 2^{2r/3}$ and $M = D = 2^{r/3}$, which require less data than the first algorithm.

Theorem 10.4. *Given (e, N) and D consecutive values $w_1, \cdots, w_D$, there exists an algorithm which retrieves one of the values $x_0, \cdots, x_{D-1}$ in time T, by using M random access memory such that $TM^2D^2 = O(2^{2r})$ with $D^2 \leq T \leq 2^r$.*

Proof. As before, we detail the algorithm and then its complexity.
Algorithm. For sake of clarity we first describe the algorithm in the case of a single Hellman's table. Let f be the function defined by $f(x) = \text{LSB}_r(x^e \mod N)$ where $\text{LSB}_r(x)$ represents the r least significant bits of x. The preprocessing phase consists in computing for m random different values $x_0^1, \cdots, x_0^m$ the values $f^t(x_0^1), \cdots, f^t(x_0^m)$ with $m, t \in \mathbb{N}$ and where f^t means that the function f is iterated t times. The construction of a hash table containing the $f^t(x_0^i)$ as keys and the x_0^i as associated values, for $i \in \{1, \cdots, m\}$, concludes this phase.
The algorithm in online phase works as follows:

1. One selects a known value w_j for $j > 0$.

2. One considers only the r least significant bits of w_j, denoted by z_j.

3. For $i \in \{0, \cdots, t\}$, one tests if $f^i(z_j)$ is a key of the hash function. If the $t+1$ tests fail, one selects the next known value and restarts the algorithm.

4. If a test succeeds, denoting the associated value x_0^c, one has:

$$f^{t-i}(x_0^c) = z_j = f(\underbrace{f^{t-i-1}(x_0^c)}_{X})$$

   X is a value of size r that corresponds with high probability to the hidden part of the generator at the previous iteration. A simple verification consists of the computation of the value $X^e \mod N$.

Figure 10.2 gives an overview of the algorithm by manipulating the hash table and using the function f.

Now a set of tables covers a larger fraction of the possible output values and consequently, the online phase need less data. Each table requires a specific function and, in order to cover different independent output values, the functions need to be independent. In [Hel80], Hellman rigorously calculated a lower bound on the expected coverage of images by a *single* table which is essentially the same analysis we did in the previous algorithm. However, the analysis for the full scheme (with *many* tables) is highly heuristic and is based on the unjustifiable assumption that many simple variants of f are independent of each other. Fiat and Naor in [FN99] propose to use k-wise independent functions in order to propose an algorithm to invert *any* function, while Hellman assumes that the function is random. In order to replace the heuristic, one could think of using independent functions for each table by computing $g_i = h_i \circ f$, where $\{h_i\}_i$ is a family of k-wise independent functions. The main drawback is that the number of such functions we need is exponential and it is not easy to construct such functions. Here, we want to avoid Hellman heuristic while similar heuristic could be made. For instance, we could define many functions by considering any r bits among the $n-r$ output bits which will give us $\binom{n-r}{r}$ different functions. However, many functions will have the same subset of bits and we cannot assume independence between them. The analysis of the algorithm is based on the following hypothesis:

Assumption 10.2. *Denoting $f(x) = \text{LSB}_r(x^e \mod N)$, the distribution of $f(x)$ for random r''-bit integers ($r'' \geq r$) x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in $[0, 2^r)$.*

Instead of using a single table, one thus uses $\ell = t/D$ hash tables of size mt (assuming that $t > D$). First, one has to find which table covers the output value. Then one applies the algorithm described above. Consequently, the search of the table requires to look for each value in all tables in parallel.

Complexity. We cover a fraction $mt/2^r$ of the output values with one table. Now, using ℓ tables, we want to prove that the number of output values we cover is $mt\ell$. To prove such result, we have to solve the independence problem, namely that to describe independent functions for each table so that we are still able to invert f. Our idea is to use the fact that f is a random function or that its outputs are indistinguishable from the uniform distribution (Assumption 10.2). By using ℓ random and independent values $z_i \in [0, 2^{r'})$, we can define ℓ functions as $g_i(x) = f(x + z_i \cdot 2^r)$ for $i \in \{1, \cdots, \ell\}$. We claim that this set of functions is independent, otherwise assumption 10.2 will be wrong for $r'' = r + r'$.

Value	Hellman's Matrix for our algorithm							Key
$\boldsymbol{x_0^1}$	$\xrightarrow{f}$	$f(x_0^1)$	$\xrightarrow{f}$	$\cdots$	$\xrightarrow{f}$	$f^{t-1}(x_0^1)$	$\xrightarrow{f}$	$\boldsymbol{f^t(x_0^1)}$
$\vdots$								$\vdots$
$\boldsymbol{x_0^c}$	$\underbrace{\xrightarrow{f}\cdots\xrightarrow{f} X}_{\text{step 4}}$		$\xrightarrow{f}$		$\underbrace{z_j \xrightarrow{f}\cdots\xrightarrow{f}}_{\text{step 3}}$			$\boldsymbol{f^i(z_j)}$
$\vdots$								$\vdots$
$\boldsymbol{x_0^m}$	$\xrightarrow{f}$	$f(x_0^m)$	$\xrightarrow{f}$	$\cdots$	$\xrightarrow{f}$	$f^{t-1}(x_0^m)$	$\xrightarrow{f}$	$\boldsymbol{f^t(x_0^m)}$

Figure 10.2: Computation of our algorithm using a hash table

The number of different values in a single table can be estimated as follows (the end value of each chain is not counted):

$$E(\#\{f^j(x_0^i), 1 \leq i \leq m, 0 \leq j < m\}) = \sum_{i=1}^{m}\sum_{j=0}^{t-1} Pr[A_{i,j}]$$

where $A_{i,j}$ the event $[f^j(x_0^i) \notin \{f^{j'}(x_0^{i'}), i' < i \text{ or } j' < j\}]$. Note that $A_{i,j} \subseteq A_{i,j-1}$ (since $f^j(x_0^i) = f^{j-1}(x_0^i)$). Moreover, we have the following property:

$$Pr[A_{i,j}|A_{i,j-1}] \geq 1 - \frac{it}{2^r} \Rightarrow Pr[A_{i,j}] \geq \big(1 - \frac{it}{2^r}\big)^{j+1}$$

Hence, the probability p that the value we search is in one Hellman's table is greater than $2^{-r}\sum_{i=1}^{m}\sum_{j=0}^{t-1}(1-\frac{it}{2^r})^{j+1}$. Now the probability p' that the value we search is in one of the ℓ Hellman's tables is greater than $1-\big(1-2^{-r}\sum_{i=1}^{m}\sum_{j=0}^{t-1}(1-\frac{it}{2^r})^{j+1}\big)^{\ell}$ and $p' \approx 2^{-r}mt\ell$ if $mt^2 \ll 2^r$. We clearly have $M = O(m\ell)$, $T = O(Dt\ell)$ and $P = O(mt\ell)$. For $\ell = t/D$, we obtain the tradeoff $TM^2D^2 = O(2^{2r})$ with $D^2 \leq T \leq 2^r$.

■

10.4 Inverting RSA for Small Plaintext Problem

By using one of the previous algorithms, one knows the value of a hidden part of the generator denoted x_i for $i \geq 0$. We now present two different ways to invert the Micali-Schnorr generator, i.e to retrieve the secret seed x_0.

Description of the problem. Let (e, N) be an RSA public key with N of size n and an integer $r \leq n$. Given (N, e, r) and $y = x^e \bmod N$ for $x \in [0, 2^r)$, the problem consists in recovering x.

Remark. This problem is well-known to be solvable in polynomial time when $r \leq n/e$ since as before the equality holds over the integers.

10.4.1 Multipoint evaluation of univariate polynomials

In our case, it is clear that using the multipoint evaluation of univariate polynomials (see Section 8.2.2 for some recalls), will lead to retrieve the seed. For example suppose that we

know the value of v_i and want to retrieve the value of x_{i-1} of the generator. That can be done by multipoint evaluating the polynomial of degree $e(2^{r/2}+1)$:

$$P(X) = (X^e - v_i)((X+1)^e - v_i)((X+2)^e - v_i)...((X+2^{r/2})^e - v_i) \mod N$$

on the points $k \cdot 2^{r/2}$ for $k = 0, \cdots, 2^{r/2}$ in order to find k_c such that $P(k_c \cdot 2^{r/2}) = 0 \mod N$. Then, one searches the value of x_{i-1} on the form $k_c \cdot 2^{r/2} + \ell$ for $\ell = 0, \cdots, 2^{r/2}$. This technique requires $\tilde{O}(e \cdot 2^{r/2})$ operations in $\mathbb{Z}_N$ and one has to store the first tree, i.e. $2^{r/2}$ polynomials.

Remark. This algorithm can be applied to attack the RSA encryption system when used to encrypt a short secret key of a symmetric cipher. Our algorithm is slightly less efficient than the one in [BJN00] but it always succeeds (whereas recovering a 40-bit plaintext for instance is successful only with probability 0.39 in [BJN00]).

10.4.2 Coppersmith's method

Another technique is based on the well-known Coppersmith's method for the case of a modular univariate polynomial (see Section 9.2 for some recalls). In our case, starting from the equation $x_{i-1}^e = v_i \mod N$, we can define the following modular univariate polynomial f as $f(x) = x^e - v_i \mod N$. The value x_{i-1} represents a small modular root of this polynomial. However, our root of size r is not enough small for this technique which requires the root to be less than $N^{1/e}$, i.e $r < n/e$ (see [Cop96b]). To circumvent this problem, one can guess j bits of x in order to have $r - j < n/e$ and then apply Coppersmith's method for each guess. Instead of f, one uses the polynomial g of degree e:

$$g(x) = (\lambda + x)^e - v_i \mod N$$

with λ the guessed value of j bits. The truncated value of x_{i-1} denoted by x_{i-1}^{tr} is a small modular root of g. Its degree being the same, the asymptotic condition on the size of the root remains the same. We refer to Chapter 9 and more precisely to Section 9.3.1 for more details.

10.4.3 Splitting probabilities for integers

The last reversing is based on the attack described by Boneh, Joux and Nguyen in [BJN00][1] and can be viewed as a meet-in-the-middle method. They use the fact that a relatively small integer can often be expressed as a product of much smaller integers.

One starts from the equation $x_{i-1}^e = v_i \mod N$ where v_i is known by the attacker ; its goal is to retrieve x_{i-1} of size r. Suppose that $x_{i-1} = AB$ with $A < 2^\alpha$ and $B < 2^\beta$ Then:

$$\frac{v_i}{B^e} = A^e \mod N$$

By building a table of size 2^α containing all the possible values of $A^e \mod N$, one just has to test for each possible value of B if $v_i/B^e \mod N$ is in the table. Any collision will reveal the value x_{i-1}.
Using the tradeoff $\alpha = \beta = r/2$, this attack requires $2^{r/2}r$ bits of memory and the computation of $2 \cdot 2^{r/2}$ modular exponentiations. The success probability of this attack is related to the famous *Erdös multiplication table problem.* Given an integer N, let $m(N)$ denote

[1] It is worth noting that this idea was already mentioned in Micali-Schnorr paper and attributed to a personal communication by Pollard.

the number of integers of the form ab with $1 \leq a, b \leq N$. Erdös proved [Erd55, Erd60] that $\lim_{N\to+\infty} m(N)/N^2 = 0$ and these products do not take up a positive proportion of the integers up to N^2 (see also [For08]). In particular, the success probability decreases with r and since i is a large value, one cannot expect retrieve the seed by using only with this attack.

10.5 Proof of Theorem 10.1 and application

In this section, we want to prove some good properties of the k least significant bits of the function $x^e \bmod N$. More precisely, to prove Theorem 10.1, we have to estimate the statistical distance between the function $\text{LSB}_k(x^e \bmod N)$ for x randomly chosen in $\mathbb{Z}/M\mathbb{Z}$ and the uniform distribution modulo 2^k. In function of the values of N, e, k and M, we will be able to show (or not) the statistically indistinguishability of these two probability ensembles. More precisely the ratio between M and N is crucial in our analysis and we propose two different techniques for estimating the statistical distance. The first one is meaningful when $M \gg \sqrt{N}$ (see Theorem 10.5) and the second one is useful for $M \ll \sqrt{N}$ (see Theorem 10.6).

10.5.1 Preliminaries

Statistical Distance

We will consider the *collision probability* of a random variable X on a finite set S, defined as:

$$\text{Col}(X) = \sum_{s\in S} \Pr[X = s]^2.$$

The link between the statistical distance, as defined in Chapter 5 and the collision probability is given by the following lemma proven in [Sho06]:

Lemma 10.2. *Let X be a random variable with values in a finite set S of size m. If X has collision probability β and distance δ from uniform on S, then:*

$$\delta \leq \frac{1}{2}\sqrt{m\beta - 1}.$$

Exponential Sums

Our proof of Theorem 10.1 relies on exponential sums in $\mathbb{Z}/m\mathbb{Z}$, so we first fix some notations and recall useful standard results. For any integer m, we denote by $\mathbf{e}_m$ the additive character $\mathbb{Z}/m\mathbb{Z} \to \mathbb{C}^*$ given by $\mathbf{e}_m(x) = \exp(2i\pi x/m)$. The following results hold.

Proposition 10.1 (Orthogonality)**.** *For all $x \in \mathbb{Z}/m\mathbb{Z}$, we have:*

$$\sum_{c=0}^{m-1} \mathbf{e}_m(cx) = \begin{cases} 0 & \text{if } c \not\equiv 0 \pmod m, \\ m & \text{if } c \equiv 0 \pmod m. \end{cases}$$

Lemma 10.3 ([Vin54, Problem 11.c])**.** *For any modulus[2] $m \geq 60$ and any non negative integers h, k, we have:*

$$\sum_{c=1}^{m-1} \left| \sum_{x=k}^{k+h} \mathbf{e}_m(cx) \right| \leq (m-1)\log m.$$

[2] We assume that this bound on m holds for all moduli involved in our computations below, i.e. $p, q > 60$, which is of course satisfied for all RSA moduli in practice.

Lemma 10.4 (Weil [LN96]). *Consider a prime modulus p. For all polynomials $g(X), h(X) \in \mathbb{F}_p[X]$ such that the rational function $f(X) = h(X)/g(X)$ is not constant on $\mathbb{F}_p$, the bound:*

$$\left| \sum_{\substack{x \in \mathbb{F}_p \\ g(x) \neq 0}} \mathbf{e}_p\big(f(x)\big) \right| \leq \big(\max(\deg g, \deg h) + v - 1\big) \cdot p^{1/2}$$

holds, with v the number of distinct zeros of $g(X)$ in the algebraic closure of $\mathbb{F}_p$.

10.5.2 First Bound when $M \gg \sqrt{N}$

The first technique we propose uses mostly the technical lemmas on exponential sums. We obtain a bound on the statistical distance which is negligible when the parameter M is sufficiently larger than $\sqrt{N}$. With this first analysis we cannot hope to approach the optimal bound, i.e. $M \gg N^{1/e}$ for $e \geq 3$. However its advantage may lie in providing concrete values of M and k for cryptographic sizes of modulus N. It is thus an useful bound for our applications.

Theorem 10.5. *Let $N = pq$ be a balanced RSA modulus (i.e. $N = pq$ for primes p, q such that $60 < q < p < 2q$), e the public exponent and $M < N$ an integer such that $M \gg \sqrt{N}$. Then the random variable $X = \text{LSB}_k(x^e \bmod N)$ for x randomly chosen in $\mathbb{Z}/M\mathbb{Z}$ is δ-statistically close to uniform with:*

$$\delta = \sqrt{\frac{2^k}{N}} + \frac{2^{k/2} e^2 \sqrt{N} \log^{3/2} N}{M}.$$

Proof. The values of the random variable X are taken in $[0, 2^k)$ with the following distribution: x is chosen uniformly at random in $\mathbb{Z}/M\mathbb{Z}$ and we output $f(x) = \text{LSB}_k(x^e \bmod N)$. We are interested in bounding the collision probability of this random variable. By denoting $K = \left\lfloor \frac{N-1}{2^k} \right\rfloor$, we can evaluate this probability using the orthogonality property of additive characters (Prop. 10.1):

$$\begin{aligned}
\text{Col}(X) =& \frac{1}{M^2} \times \left| \{(x, y) \in [0, M-1]^2 \,|\, \exists u \in [\![-K, K]\!], x^e - y^e = 2^k \cdot u \bmod N\} \right|, \\
\leq& \frac{2}{M^2 N} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \sum_{u=0}^{K} \sum_{a=0}^{N-1} \mathbf{e}_N(a(x^e - y^e - 2^k \cdot u)).
\end{aligned}$$

We now define by B the value $\max_a |S(a, M)|$ where $S(a, M) = \sum_{0 \leq x < M} \mathbf{e}_N(ax^e)$ to uniformly bound $\sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \mathbf{e}_N(ax^e - ay^e) = S(a, M)\overline{S(a, M)} = |S(a, M)|^2$. The contribution of $a = 0$ is exactly $2(K+1)/N$ and if we put it aside we get:

$$\text{Col}(X) \leq \frac{2(K+1)}{N} + \frac{2}{M^2 N} \sum_{a=1}^{N-1} |S(a, M)|^2 \left| \sum_{u=0}^{K} \mathbf{e}_N(-a2^k \cdot u) \right|.$$

The probability collision can be bounded using B and Lemma 10.3 since the function $a \in \mathbb{Z}/N\mathbb{Z} \setminus \{0\} \to 2^k a \in \mathbb{Z}/N\mathbb{Z} \setminus \{0\}$ is a bijection:

$$\text{Col}(X) \leq \frac{2(K+1)}{N} + \frac{2}{M^2 N} \left(\max_{1 \leq a \leq N-1} |S(a, M)|^2 \right) \sum_{a=1}^{N-1} \left| \sum_{u=0}^{K} \mathbf{e}_N(-a2^k \cdot u) \right|,$$

$$\text{Col}(X) \leq \frac{2(K+1)}{N} + \frac{2}{M^2 N} B^2 \cdot N \log N. \tag{10.2}$$

It remains to bound B for all $a \in \mathbb{Z}/N\mathbb{Z} \setminus \{0\}$. The incomplete exponential sum $S(a, M)$ can be expressed using the complete one as follow:

$$\begin{aligned} S(a,M) = \sum_{x<M} \mathbf{e}_N(ax^e) &= \sum_{x\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N(ax^e) \cdot \big[x \in [\![0, M-1]\!]\big], \\ &= \sum_{x\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N(ax^e) \frac{1}{N} \sum_{m=0}^{M-1} \sum_{b\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N(b(x-m)), \\ &= \frac{1}{N} \sum_{b\in\mathbb{Z}/N\mathbb{Z}} \sum_{m=0}^{M-1} \mathbf{e}_N(-bm) \sum_{x\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N(ax^e + bx). \end{aligned}$$

where $[\cdot]$ is the usual Iverson bracket notation: for a statement U, $[U] = 1$ is U is true and 0 otherwise. If we pick integers u, v such that $up + vq = 1$, we see that the sum in x decomposes as:

$$\begin{aligned} \sum_{x\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N(ax^e + bx) &= \sum_{x\in\mathbb{Z}/N\mathbb{Z}} \mathbf{e}_N\big((up+vq)\cdot(ax^e+bx)\big) \\ &= \sum_{x_p\in\mathbb{F}_p} \mathbf{e}_p\big(vg_{a,b}(x_p) \bmod p\big) \sum_{x_q\in\mathbb{F}_q} \mathbf{e}_q\big(ug_{a,b}(x_q) \bmod q\big) \end{aligned}$$

where the function $g_{a,b}$ is given by $g_{a,b}(x) = ax^e + bx$. Now if $a \neq 0 \bmod p$, $vg_{a,b}$ is a non constant function in $\mathbb{F}_p$, so Lemma 10.4 ensures:

$$|T_p| = \left| \sum_{x_p\in\mathbb{F}_p} \mathbf{e}_p(vg_{a,b}(x_p) \bmod p) \right| \leq e\sqrt{p}.$$

On the other hand, if $a = 0 \bmod p$, we have:

$$\sum_{x_p\in\mathbb{F}_p} \mathbf{e}_p(vg_{a,b}(x_p) \bmod p) = \sum_{x_p\in\mathbb{F}_p} \mathbf{e}_p(vbx) = \begin{cases} p & \text{if } b = 0 \bmod p, \\ 0 & \text{otherwise.} \end{cases}$$

A corresponding result holds for $T_q = \sum_{x_q\in\mathbb{F}_q} \mathbf{e}_q(ug_{a,b}(x_q) \bmod q)$ and as a result, to bound $|S(a, M)|$, we have to separate the case when a is invertible modulo N from the case where it is a multiple of p or q. When a is invertible modulo N, we directly have:

$$\begin{aligned} |S(a,M)| &\leq \frac{1}{N} \sum_{b\in\mathbb{Z}/N\mathbb{Z}} \left| \sum_{m=0}^{M-1} \mathbf{e}_N(-bm) \right| \cdot |T_p| \cdot |T_q|, \\ &\leq \frac{1}{N} \sum_{b\in\mathbb{Z}/N\mathbb{Z}} \left| \sum_{m=0}^{M-1} \mathbf{e}_N(-bm) \right| \cdot e\sqrt{p} \cdot e\sqrt{q} \leq \frac{1}{N} e^2 \sqrt{N} \cdot N \log N \end{aligned}$$

by Lemma 10.3. On the other hand, assume that a is a multiple of p. We have:

$$\begin{aligned} |S(a,M)| &= \left| \frac{p}{N} \sum_{\substack{b\in\mathbb{Z}/N\mathbb{Z} \\ b=0 \bmod p}} \sum_{m=0}^{M-1} \mathbf{e}_N(-bm) \sum_{x_q\in\mathbb{F}_q} \mathbf{e}_q\big(ug_{a,b}(x_q) \bmod q\big) \right| \\ &\leq \frac{1}{q} \sum_{b'=0}^{q-1} \left| \sum_{m=0}^{M-1} \mathbf{e}_N(-bm) \right| \cdot e\sqrt{q} \leq e\sqrt{q} \log q < e^2 \sqrt{N} \log N, \end{aligned}$$

by applying Lemma 10.3. the same bound holds when a is a multiple of q, so that $B \leq e^2\sqrt{N}\log N$. Together with (10.2), we obtain:

$$\mathrm{Col}(X) \leq \frac{2(K+1)}{N} + \frac{2e^4 N \log^3 N}{M^2}.$$

Finally Lemma 10.2 provides a bound on the statistical distance $\Delta_1(X)$:

$$\Delta_1(X) \leq \sqrt{\frac{2^k(K+1)}{N} - 1} + \frac{2^{k/2}e^2\sqrt{N}\log^{3/2} N}{M},$$

and to conclude the proof, it is easy to show that $\left|\frac{K+1}{N} - 1/2^k\right| \leq 1/N$.

■

10.5.3 Second Bound when $M \ll \sqrt{N}$

Here we treat the case where M is smaller than $\sqrt{N}$, which will be interesting to approach the optimal bound, *i.e.* $M \gg N^{1/e}$. Even if the following lemma and corollaries do not require anything on the size of M (except that it is less than N obviously), the bounds we find for $\Delta_1(X)$ are only interesting for small values of M, meaning $M \ll \sqrt{N}$.

Theorem 10.6. *Let Let N, e, X be as in Theorem 10.5 and $M < N$ an integer such that $M \ll \sqrt{N}$. Then X is δ-statistically close to uniform with:*

$$\delta = \sqrt{\frac{2^k}{N}} + 2^{k/2}\log^{3/2} N\left(\frac{1}{M} + \frac{N}{M^e}\right)^{\frac{1}{2e(e-1)+1}} + \frac{2^{k/2}e\log^{3/2} N}{M}.$$

Proof. This result is based on a more recent result proved by Wooley, which provides another evaluation of the exponential sum $S(a, M)$. We give here a specific version adapted to our case:

Theorem 10.7 (Wooley, [Woo12])**.** *Let e be an integer with $e \geq 2$, and let $a/N \in \mathbb{R}$. Suppose that, for some $c \in \mathbb{Z}$ and $N \in \mathbb{N}$ with $\gcd(c, N) = 1$, one has $|a/N - c/N| \leq N^{-2}$ and $N \leq M^e$. Then one has:*

$$\sum_{1 \leq x \leq M} \mathbf{e}_N(ax^e) \ll M^{1+\varepsilon}(N^{-1} + M^{-1} + N \cdot M^{-e})^{\sigma(e)} \quad \textit{where} \quad \sigma(e)^{-1} = 2e(e-1).$$

According to [Woo12], the factor M^ε may be replaced by $\log(2M)$, if one increases $\sigma(e)^{-1}$ from $2e(e-1)$ to $2e^2 - 2e + 1$. For sake of simplicity, we bound $\log(2M)$ by $\log N$ (with the weak assumption that $M \leq N/2$) and we neglect the term $1/N$ since it is negligible compared to $\min(M^{-1}, N \cdot M^{-e})$. Thus we obtain:

$$|S(a, M)| \ll M \log N (M^{-1} + N \cdot M^{-e})^{\frac{1}{2e(e-1)+1}}.$$

Note that this bound is correct for $a \in (\mathbb{Z}/N\mathbb{Z})^*$ and because it is meaningful when $M \ll \sqrt{N}$, one cannot bound as before the value $|S(a, M)|$ for $a \notin (\mathbb{Z}/N\mathbb{Z})^*$. Starting from:

$$\mathrm{Col}(X) \leq \frac{2(K+1)}{N} + \frac{2}{M^2 N}\sum_{a=1}^{N-1} |S(a, M)|^2 \left|\sum_{u=0}^{K} \mathbf{e}_N(-a2^k \cdot u)\right|,$$

we decompose the sum in a as:

$$\sum_{a=1}^{N-1} |S(a, M)|^2 \left|\sum_{u=0}^{K} \mathbf{e}_N(-a2^k \cdot u)\right| = S^* + S_p + S_q$$

with S^* the sum in $a \in (\mathbb{Z}/N\mathbb{Z})^*$ and S_p (resp. S_q) the one in $a \in \mathbb{Z}/N\mathbb{Z} \setminus \{0\}$ such that $a = 0 \bmod p$ (resp. $a = 0 \bmod q$). The sum S^* is bounded using Theorem 10.7 and Lemma 10.3, whereas we treat S_p (and similarly S_q) using an intermediate result from the previous proof and Lemma 10.3, i.e.:

$$S^* \leq M^2 \log^2 N (M^{-1} + N \cdot M^{-e})^{\frac{2}{2e(e-1)+1}} \cdot N \log N,$$
$$S_p \leq e^2 q \log^2 q \cdot \sum_{a_p \in \mathbb{F}_q^*} \left| \sum_{u=0}^{K} \mathbf{e}_q(-a_p 2^k \cdot u) \right| \leq e^2 q^2 \log^3 q \leq e^2 N \log^3 N.$$

We thus have:

$$\mathrm{Col}(X) \leq \frac{2(K+1)}{N} + 2\log^3 N (M^{-1} + N \cdot M^{-e})^{\frac{2}{2e(e-1)+1}} + \frac{4e^2 \log^3 N}{M^2}$$
$$\Delta_1(X) \leq \sqrt{\frac{2^k}{N}} + 2^{k/2} \log^{3/2} N \left(\frac{1}{M} + \frac{N}{M^e} \right)^{\frac{1}{2e(e-1)+1}} + \frac{2^{k/2} e \log^{3/2} N}{M}$$

■

To conclude, we extend ours theorems to the most significant bits case.

Corollary 10.1. *Let $N = pq$ the product of two large prime integers and M an integer less than N. The results from Theorem 10.5 and Theorem 10.6 on the statistical distance between $\mathrm{LSB}_k(x^e \bmod N)$ for x randomly chosen in $\mathbb{Z}_M$ and U_k are still valid for $\mathrm{MSB}_k(x^e \bmod N)$.*

Proof. Indeed, the single difference appears in the evaluation of $\mathrm{Col}(X)$:

$$\begin{aligned} \mathrm{Col}(X) &= \frac{1}{M^2} \times \left| \{(x,y) \in [0, M-1]^2 \mid \exists u \leq K, x^e - y^e = u \bmod N\} \right|, \\ &\leq \frac{2}{M^2 N} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \sum_{u=0}^{K} \sum_{a=0}^{N-1} e_N(a(x^e - y^e - u)), \end{aligned}$$

with K still equal to $\lfloor \frac{N-1}{2^k} \rfloor$. Since

$$\sum_{a=1}^{N-1} \left| \sum_{u=0}^{K} e_N(-a 2^k \cdot u) \right| = \sum_{a=1}^{N-1} \left| \sum_{u=0}^{K} e_N(-a \cdot u) \right|,$$

the rest of the proof remains the same.

■

10.5.4 Application to the Micali-Schnorr generator

We do not contradict Assumption 10.1 since our theorems give upper bounds on δ but note that it cannot be true if one does not restrain the tests to polynomial-time ones only because of the lack of entropy in input. Micali and Schnorr have proposed the parameters $r = 2n/e$ and thus $k = n(1-2/e)$ which are very aggressive parameters in order to increase the efficiency of the generator.

Theorem 10.5 is useful to define the sizes of parameters k and r such that the statistical distance is bounded as desired. Corollary 10.2 consists in determining the minimal size of the input in order to have an indistinguishable output from the uniform distribution. This is really interesting to note that when N tends to infinity, this bound tends to $2/3$. In other words we cannot expect to have a positive result of indistinguishability according to our results if one outputs more than $(\log N)/3$ of the least significant bits asymptotically.

Corollary 10.2. *Let (e, N) a RSA public key with e small compared to $\log N$ and d a security parameter such that $\delta < 2^{-d}$. Let $\alpha \in (0,1)$ such that Micali-Schnorr pseudo-random number generator outputs the $(1-\alpha)\log N$ least significant bits at one iteration. This output is indistinguishable from the uniform distribution on $\{0,1\}^{(1-\alpha)\log N}$ if*

$$\alpha > 2/3 + \frac{2d + 4\log e}{3\log N} + \frac{\log\log N}{\log N}.$$

Proof. By denoting $M = N^\alpha$ with $\alpha < 1$ and $2^k \simeq N^{1-\alpha}$, we obtain from Theorem 10.5:

$$\delta \leq N^{-\alpha/2} + N^{1-\frac{3}{2}\alpha} e^2 \sqrt{N} \log^{3/2} N.$$

Since $-\alpha/2 < 1 - \frac{3}{2}\alpha$ is equivalent to $\alpha < 1$ and $e^2\sqrt{N}\log^{3/2} N$ is quite large for cryptographic parameters, we will neglect the first term. We bound the statistical distance by 2^{-d} with d a security parameter and obtain the following condition on α:

$$\alpha > 2/3 + \frac{2d + 4\log e}{3\log N} + \frac{\log\log N}{\log N}.$$

■

As a concrete example, for $N = 2^{1024}$, $e = 3$ and $d = 80$, that gives an input greater than 747 bits (and thus an output lesser than 277 bits). Note that we study a single iteration of the generator as in [FS01] for example, the consideration of two or more consecutive outputs is a more difficult task. Finally remark that Theorem 10.6 is useless for this application because of the necessarily link between k and M: if $M = N^\alpha$ then $2^k \simeq N^{1-\alpha}$. For $M \ll \sqrt{N}$, there is not enough entropy to prove the indistinguishability.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABBD13] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 1217–1230. ACM, 2013. (Cited on page 63.)

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In Jr. et al. [JcKKP03], pages 260–275. (Cited on page 21.)

[ABJ+13] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *FMCAD*, pages 1–17. IEEE, 2013. (Cited on page 48.)

[AFG+14] Diego F. Aranha, Pierre-Alain Fouque, Benoit Gérard, Jean-Gabriel Kammerer, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Glv/gls decomposition, power analysis, and attacks on ecdsa signatures with single-bit nonce bias. In *ASIACRYPT*, 2014. *to appear*. (Cited on pages v, 13, 20, and 80.)

[AFQ+14] Diego F. Aranha, Pierre-Alain Fouque, Chen Qian, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Binary elligator squared. Cryptology ePrint Archive, Report 2014/486, 2014. http://eprint.iacr.org/. (Cited on pages v, 14, 19, and 101.)

[AG] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. http://code.google.com/p/relic-toolkit/. (Cited on pages 88 and 112.)

[ÁH14] Erika Ábrahám and Klaus Havelund, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*. Springer, 2014. (Cited on pages 60 and 181.)

[AK97] Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997. (Cited on pages 21 and 61.)

[ALH10] Diego F. Aranha, Julio López, and Darrel Hankerson. Efficient software implementation of binary field arithmetic using vector instruction sets. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of

Lecture Notes in Computer Science, pages 144–161. Springer, 2010. (Cited on page 111.)

[ANS11] ANSSI. Publication d'un paramétrage de courbe elliptique visant des applications de passeport électronique et de l'administration électronique française, November 2011. (Cited on page 102.)

[AScKK05] Onur Aciiçmez, Werner Schindler, and Çetin Kaya Koç. Improving brumley and boneh timing attack on unprotected ssl implementations. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 139–146. ACM, 2005. (Cited on page 27.)

[Bab95] Steve Babbage. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *IEE Conference Publication - European Convention on Security and Detection*, volume 408, 1995. (Cited on page 154.)

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM. (Cited on page 89.)

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005. (Cited on page 27.)

[BBBP13] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, and Gerardo Pelosi. A fault induction technique based on voltage underfeeding with application to attacks against aes and rsa. *Journal of Systems and Software*, 86(7):1864–1878, 2013. (Cited on page 21.)

[BBPS11] Alessandro Barenghi, Guido Bertoni, Andrea Palomba, and Ruggero Susella. A novel fault attack against ecdsa. In *HOST*, pages 161–166. IEEE Computer Society, 2011. (Cited on page 81.)

[BBS86] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986. (Cited on pages 8, 115, and 155.)

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2006. (Cited on page 154.)

[BCC+13] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. Factoring RSA keys from certified smart cards: Coppersmith in the wild. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 341–360. Springer, 2013. (Cited on page 7.)

[BCG+13] Gilles Barthe, Juan Manuel Crespo, Sumit Gulwani, César Kunz, and Mark Marron. From relational verification to simd loop synthesis. In Alex Nicolau, Xiaowei Shen, Saman P. Amarasinghe, and Richard W. Vuduc, editors, *PPOPP*, pages 123–134. ACM, 2013. (Cited on page 60.)

[BCHL13] Joppe W. Bos, Craig Costello, Hüseyin Hisil, and Kristin Lauter. High-performance scalar multiplication using 8-dimensional glv/gls decomposition. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 331–348. Springer, 2013. (Cited on page 80.)

[BCI+09] Eric Brier, Jean-Sebastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. Cryptology ePrint Archive, Report 2009/340, 2009. http://eprint.iacr.org/. Full version of [BCI+10]. (Cited on pages 103 and 104.)

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2010. (Cited on pages 102 and 175.)

[BD92] Jørgen Brandt and Ivan Damgård. On generation of probable primes by incremental search. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 358–370. Springer, 1992. (Cited on pages 127 and 128.)

[BD00] Dan Boneh and Glenn Durfee. Cryptanalysis of rsa with private key d less than $n^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339–1349, 2000. (Cited on page 133.)

[BDF+14a] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Making RSA-PSS provably secure against non-random faults. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2014. (Cited on pages v, 12, 19, and 61.)

[BDF+14b] Gilles Barthe, Francois Dupressoir, Pierre-Alain Fouque, Benjamin Gregoire, and Jean-Christophe Zapalowicz. Synthesis of fault attacks on cryptographic implementations. Cryptology ePrint Archive, Report 2014/436, 2014. http://eprint.iacr.org/. (Cited on pages v, 11, 13, 19, and 45.)

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997. (Cited on pages 21, 25, and 45.)

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001. (Cited on page 21.)

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012. (Cited on pages 103 and 113.)

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006. (Cited on pages 103 and 113.)

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011. (Cited on pages 54 and 63.)

[BGM97] Mihir Bellare, Shafi Goldwasser, and Daniele Micciancio. "pseudo-random" number generation within cryptographic algorithms: The dds case. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 1997. (Cited on page 120.)

[BGPGS03] Simon R. Blackburn, Domingo Gomez-Perez, Jaime Gutierrez, and Igor Shparlinski. Predicting the inversive generator. In Kenneth G. Paterson, editor, *IMA Int. Conf.*, volume 2898 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2003. (Cited on pages 131 and 132.)

[BGPGS05] Simon R. Blackburn, Domingo Gomez-Perez, Jaime Gutierrez, and Igor Shparlinski. Predicting nonlinear pseudorandom number generators. *Math. Comput.*, 74(251):1471–1494, 2005. (Cited on pages 131, 132, 136, 142, 143, and 144.)

[BGPGS06] Simon R. Blackburn, Domingo Gomez-Perez, Jaime Gutierrez, and Igor Shparlinski. Reconstructing noisy polynomial evaluation in residue rings. *J. Algorithms*, 61(2):47–59, 2006. (Cited on pages 131 and 132.)

[BH05] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 203–212. ACM, 2005. (Cited on page 115.)

[BH09] Billy Bob Brumley and Risto M. Hakala. Cache-Timing Template Attacks. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer, 2009. (Cited on page 81.)

[BHH+13] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. *IACR Cryptology ePrint Archive*, 2013:734, 2013. (Cited on page 5.)

[BHKL13a] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 967–980. ACM, 2013. (Cited on pages 102 and 113.)

[BHKL13b] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Software. http://elligator.cr.yp.to/software.html, August 2013. (Cited on page 103.)

[BHT10] Arnaud Boscher, Helena Handschuh, and Elena Trichina. Fault resistant RSA signatures: Chinese remaindering in both directions. *IACR Cryptology ePrint Archive*, 2010:38, 2010. (Cited on page 61.)

[BJ07] Aurélie Bauer and Antoine Joux. Toward a rigorous variation of coppersmith's algorithm on three variables. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2007. (Cited on page 133.)

[BJN00] Dan Boneh, Antoine Joux, and Phong Q. Nguyen. Why textbook elgamal and rsa encryption are insecure. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2000. (Cited on page 161.)

[BJPW13] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal collision correlation attack on elliptic curves. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 553–570. Springer, 2013. (Cited on page 81.)

[BKNS10] Joppe W. Bos, Thorsten Kleinjung, Ruben Niederhagen, and Peter Schwabe. Ecc2k-130 on cell cpus. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2010. (Cited on page 112.)

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003. (Cited on page 81.)

[Ble00] Daniel Bleichenbacher. On the generation of one-time keys in DL signature schemes. *Presentation at IEEE P1363 Working Group meeting*, 2000. (Cited on pages 81, 83, 88, and 120.)

[BLMT11] Mathieu Baudet, David Lubicz, Julien Micolod, and André Tassiaux. On the security of oscillator-based random number generators. *J. Cryptology*, 24(2):398–425, 2011. (Cited on page 8.)

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984. (Cited on pages 8 and 115.)

[BM05] Johannes Blömer and Alexander May. A tool kit for finding small roots of bivariate polynomials over the integers. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2005. (Cited on page 133.)

[BMM00] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000. (Cited on page 81.)

[BN09] Billy Bob Brumley and Kaisa Nyberg. On Modular Decomposition of Integers. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 386–402. Springer, 2009. (Cited on pages 80 and 92.)

[BNNT11a] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. Modulus fault attacks against rsa-crt signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2011. (Cited on pages 28, 51, and 52.)

[BNNT11b] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. Modulus fault attacks against rsa-crt signatures. *J. Cryptographic Engineering*, 1(3):243–253, 2011. (Cited on page 52.)

[BOS03] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. A new crt-rsa algorithm secure against bellcore attacks. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 311–320. ACM, 2003. (Cited on page 25.)

[Boy89a] Joan Boyar. Inferring sequences produced by a linear congruential generator missing low-order bits. *J. Cryptology*, 1(3):177–184, 1989. (Cited on page 119.)

[Boy89b] Joan Boyar. Inferring sequences produced by pseudo-random number generators. *J. ACM*, 36(1):129–141, 1989. (Cited on page 119.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993. (Cited on page 154.)

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994. (Cited on page 154.)

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996. (Cited on page 154.)

[BR98] Mihir Bellare and Phillip Rogaway. PSS: Provably secure encoding method for digital signatures. Submission to IEEE P1363, 1998. (Cited on pages 22 and 25.)

[BR01] Mihir Bellare and Phillip Rogaway. Probabilistic signature scheme. Patent, 2001. US 6266771. (Cited on pages 22 and 25.)

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006. (Cited on page 55.)

[BRNI13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 293–310. Springer, 2013. (Cited on page 48.)

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000. (Cited on pages 154 and 158.)

[BS04] Alin Bostan and Éric Schost. On the complexities of multipoint evaluation and interpolation. *Theor. Comput. Sci.*, 329(1-3):223–235, 2004. (Cited on page 121.)

[BS05] Alin Bostan and Éric Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005. (Cited on page 121.)

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking rsa may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71. Springer, 1998. (Cited on pages 5 and 154.)

[BVZ12] Aurélie Bauer, Damien Vergnaud, and Jean-Christophe Zapalowicz. Inferring sequences produced by nonlinear pseudorandom number generators using coppersmith's methods. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 609–626. Springer, 2012. (Cited on pages v, 16, 19, and 131.)

[CCGV13] Maria Christofi, Boutheina Chetali, Louis Goubin, and David Vigilant. Formal verification of a crt-rsa implementation against fault attacks. *J. Cryptographic Engineering*, 3(3):157–167, 2013. (Cited on pages 48 and 63.)

[CELL10] Gary Chun Tak Chow, Ken Eguro, Wayne Luk, and Philip Heng Wai Leong. A karatsuba-based montgomery multiplier. In *FPL*, pages 434–437. IEEE, 2010. (Cited on pages 39, 40, and 43.)

[Cer10] Certicom Research. SEC 2: Recommended elliptic curve domain parameters, Version 2.0, January 2010. (Cited on page 102.)

[CG99] Ran Canetti and Shafi Goldwasser. An efficient *threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999. (Cited on page 61.)

[CGM+10] Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault attacks and countermeasures on vigilant's rsa-crt algorithm. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010. (Cited on page 25.)

[CH11] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. *CoRR*, abs/1108.2714, 2011. (Cited on pages 26, 34, and 37.)

[CHS14] Craig Costello, Hüseyin Hisil, and Benjamin Smith. Faster Compact Diffie-Hellman: Endomorphisms on the x-line. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2014. (Cited on page 80.)

[CJ05a] Mathieu Ciet and Marc Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Des. Codes Cryptography*, 36(1):33–43, 2005. (Cited on page 81.)

[CJ05b] Mathieu Ciet and Marc Joye. Practical fault countermeasures for Chinese remaindering based cryptosystems. In L. Breveglieri and I. Koren, editors, *FDTC*, pages 124–131, 2005. (Cited on page 25.)

[CJK+09] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. Fault attacks on rsa signatures with partially unknown messages. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 444–456. Springer, 2009. (Cited on page 25.)

[CJM+11] Jean-Sébastien Coron, Antoine Joux, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Cryptanalysis of the rsa subgroup assumption from tcc 2005. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 147–155. Springer, 2011. (Cited on page 121.)

[CM09] Jean-Sébastien Coron and Avradip Mandal. Pss is secure against random fault attacks. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 653–666. Springer, 2009. (Cited on pages 25, 26, 62, 63, and 65.)

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. (Cited on page 81.)

[CN12] Yuanmi Chen and Phong Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer, 2012. (Cited on pages 26, 34, and 121.)

[CNT10] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Fault attacks against emv signatures. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2010. (Cited on page 25.)

[Cop96a] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996. (Cited on pages 131 and 133.)

[Cop96b] Don Coppersmith. Finding a small root of a univariate modular equation. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in*

Computer Science, pages 155–165. Springer, 1996. (Cited on pages 131, 133, and 161.)

[Cor] Intel Corporation. Intel Digital Random Number Generator (DRNG). https://software.intel.com/sites/default/files/managed/4d/91/DRNG_Software_Implementation_Guide_2.0.pdf. (Cited on page 112.)

[CS05] Scott Contini and Igor Shparlinski. On stern's attack against secret truncated linear congruential generators. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 52–60. Springer, 2005. (Cited on page 119.)

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. (Cited on pages 3 and 5.)

[DN03] H. A. David and H. N. Nagaraja. *Order Statistics*. Wiley, 2003. (Cited on page 86.)

[Erd55] Paul Erdös. Some remarks on number theory. *Riveon Lematematika*, 9:45–48, 1955. (Cited on page 162.)

[Erd60] Paul Erdös. An asymptotic inequality in the theory of numbers. *Vestnik Leningrad Univ.*, 15:41–49, 1960. (Cited on page 162.)

[EW14] Hassan Eldib and Chao Wang. Synthesis of masking countermeasures against side channel attacks. In Armin Biere and Roderick Bloem, editors, *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2014. (Cited on page 48.)

[EWS14] Hassan Eldib, Chao Wang, and Patrick Schaumont. Smt-based verification of software countermeasures against side-channel attacks. In Ábrahám and Havelund [ÁH14], pages 62–77. (Cited on page 48.)

[Far11] Reza Rezaeian Farashahi. Hashing into hessian curves. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2011. (Cited on page 102.)

[FBJ+] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo. Intel AVX: New frontiers in performance improvement and energy efficiency. White paper. http://software.intel.com/. (Cited on page 110.)

[FFS+13] Reza Rezaeian Farashahi, Pierre-Alain Fouque, Igor Shparlinski, Mehdi Tibouchi, and José Felipe Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Math. Comput.*, 82(281), 2013. (Cited on pages 102, 103, and 106.)

[FGL+12] Pierre-Alain Fouque, Nicolas Guillermin, Delphine Leresteux, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Attacking rsa-crt signatures with faults on montgomery multiplication. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 447–462. Springer, 2012. (Cited on pages v, 10, 19, and 25.)

[FGL+13] Pierre-Alain Fouque, Nicolas Guillermin, Delphine Leresteux, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Attacking rsa-crt signatures with faults on montgomery multiplication. *J. Cryptographic Engineering*, 3(1):59–72, 2013. (Cited on pages v, 10, 20, and 25.)

[FHK+88] Alan M. Frieze, Johan Håstad, Ravi Kannan, J. C. Lagarias, and Adi Shamir. Reconstructing truncated integer variables satisfying linear congruences. *SIAM J. Comput.*, 17(2):262–280, 1988. (Cited on page 119.)

[Fid72] Charles M. Fiduccia. Polynomial evaluation via the division algorithm: The fast fourier transform revisited. In Patrick C. Fischer, H. Paul Zeiger, Jeffrey D. Ullman, and Arnold L. Rosenberg, editors, *STOC*, pages 88–93. ACM, 1972. (Cited on page 121.)

[FIP09] FIPS PUB 186-3. *Digital Signature Standard (DSS)*. NIST, USA, 2009. (Cited on pages 5 and 102.)

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation". (Cited on page 88.)

[FJT13] Pierre-Alain Fouque, Antoine Joux, and Mehdi Tibouchi. Injective encodings to elliptic curves. In Colin Boyd and Leonie Simpson, editors, *ACISP*, volume 7959 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2013. (Cited on page 102.)

[FLV12] Pierre-Alain Fouque, Delphine Leresteux, and Frédéric Valette. Using faults for buffer overflow effects. In Sascha Ossowski and Paola Lecca, editors, *SAC*, pages 1638–1639. ACM, 2012. (Cited on pages 21 and 55.)

[FMP03] Pierre-Alain Fouque, Gwenaëlle Martinet, and Guillaume Poupard. Attacking unbalanced rsa-crt using spa. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2003. (Cited on page 48.)

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. (Cited on pages 154 and 159.)

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2001. (Cited on page 154.)

[For08] Kevin Ford. The distribution of integers with a divisor in a given interval. *Ann. Math*, 168(2):367–433, 2008. (Cited on page 162.)

[FS01] John Friedlander and Igor Shparlinski. On the distribution of the power generator. *Math. Comput.*, 70(236):1575–1589, 2001. (Cited on page 167.)

[FT12] Pierre-Alain Fouque and Mehdi Tibouchi. Indifferentiable hashing to barreto-naehrig curves. In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT*, volume 7533 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2012. (Cited on pages 106 and 110.)

[FTZ13] Pierre-Alain Fouque, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Recovering private keys generated with weak prngs. In Martijn Stam, editor, *IMA Int. Conf.*, volume 8308 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2013. (Cited on pages v, 15, 19, and 119.)

[FVZ13] Pierre-Alain Fouque, Damien Vergnaud, and Jean-Christophe Zapalowicz. Time/memory/data tradeoffs for variants of the rsa problem. In Ding-Zhu Du and Guochuan Zhang, editors, *COCOON*, volume 7936 of *Lecture Notes in Computer Science*, pages 651–662. Springer, 2013. (Cited on pages v, 17, 19, and 153.)

[FZ14] Pierre-Alain Fouque and Jean-Christophe Zapalowicz. Statistical properties of short rsa distribution and their cryptographic applications. In *COCOON*, 2014. *to appear*. (Cited on pages v, 17, 19, and 153.)

[Gal99] Robert Gallant. Efficient multiplication on curves having an endomorphism of norm 1. Workshop on Elliptic Curve Cryptography, 1999. (Cited on page 80.)

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984. (Cited on page 5.)

[Gar59] Harvey L. Garner. The residue number system. In *IRE-AIEE-ACM '59 (Western)*, pages 146–153. ACM, 1959. (Cited on page 28.)

[GFW13] Claire Le Goues, Stephanie Forrest, and Westley Weimer. Current challenges in automatic software repair. *Software Quality Journal*, 21(3):421–443, 2013. (Cited on page 49.)

[GGI05] Domingo Gómez, Jaime Gutierrez, and Álvar Ibeas. Cryptanalysis of the quadratic generator. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *INDOCRYPT*, volume 3797 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2005. (Cited on pages 131 and 132.)

[GGI06] Domingo Gómez, Jaime Gutierrez, and Álvar Ibeas. Attacking the pollard generator. *IEEE Transactions on Information Theory*, 52(12):5518–5523, 2006. (Cited on pages 131, 132, and 147.)

[GI13] Aurore Guillevic and Sorina Ionica. Four-Dimensional GLV via the Weil Restriction. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2013. (Cited on page 80.)

[Gir06] Christophe Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006. (Cited on page 25.)

[GJTV11] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. Synthesis of loop-free programs. In Mary W. Hall and David A. Padua, editors, *PLDI*, pages 62–73. ACM, 2011. (Cited on page 48.)

[GLS11] Steven D. Galbraith, Xibin Lin, and Michael Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011. (Cited on pages 80 and 90.)

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001. (Cited on pages 80 and 93.)

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008. (Cited on page 50.)

[GNFW12] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *IEEE Trans. Software Eng.*, 38(1):54–72, 2012. (Cited on page 49.)

[Gol97] Jovan Dj. Golic. Cryptanalysis of alleged a5 stream cipher. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997. (Cited on page 154.)

[GP02] Jorge Guajardo and Christof Paar. Itoh-tsujii inversion in standard basis and its application in cryptography and codes. *Des. Codes Cryptography*, 25(2):207–216, 2002. (Cited on page 112.)

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 205–220. USENIX Association, 2012. (Cited on page 7.)

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. (Cited on pages 154, 158, and 159.)

[HG97] Nick Howgrave-Graham. Finding small roots of univariate modular equations revisited. In Michael Darnell, editor, *IMA Int. Conf.*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997. (Cited on page 133.)

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In Joseph H. Silverman, editor, *CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001. (Cited on pages 26, 34, and 133.)

[HGKEG08] Miaoqing Huang, Kris Gaj, Soonhak Kwon, and Tarek A. El-Ghazawi. An optimized hardware architecture for the montgomery multiplication algorithm. In Ronald Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2008. (Cited on pages 39, 40, 41, and 169.)

[HGS01] Nick Howgrave-Graham and Nigel P. Smart. Lattice Attacks on Digital Signature Schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001. (Cited on pages 80, 94, 97, and 120.)

[HKM09] Darrel Hankerson, Koray Karabina, and Alfred Menezes. Analyzing the galbraith-lin-scott point multiplication method for elliptic curves over binary fields. *IEEE Trans. Computers*, 58(10):1411–1420, 2009. (Cited on page 111.)

[HM09] Mathias Herrmann and Alexander May. Attacking power generators using unravelled linearization: When do we output too much? In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2009. (Cited on pages 132, 138, 145, 147, and 150.)

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Inf. Comput.*, 78(3):171–177, 1988. (Cited on page 112.)

[JcKKP03] Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2003. (Cited on pages 43 and 173.)

[JGST10] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel, editors, *ICSE (1)*, pages 215–224. ACM, 2010. (Cited on page 48.)

[JLQ99] Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *J. Cryptology*, 12(4):241–245, 1999. (Cited on page 21.)

[JM06] Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking rsa variants. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2006. (Cited on page 133.)

[JS98] Antoine Joux and Jacques Stern. Lattice reduction: A toolbox for the cryptanalyst. *J. Cryptology*, 11(3):161–185, 1998. (Cited on page 119.)

[JT12] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012. (Cited on pages 21 and 188.)

[KA96] Çetin Kaya Koç and Tolga Acar. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996. (Cited on pages 27 and 39.)

[Kal03] Burt S. Kaliski. Raising the standard for RSA signatures: RSA-PSS. CryptoBytes Technical Newsletter, February 2003. http://www.rsa.com/rsalabs/node.asp?id=2005. (Cited on pages 22 and 25.)

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. (Cited on page 7.)

[Knu99] Erik Woodward Knudsen. Elliptic scalar multiplication using point halving. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *ASIACRYPT*, volume 1716 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 1999. (Cited on page 106.)

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):pp. 203–209, 1987. (Cited on pages 5 and 75.)

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. (Cited on page 7.)

[Lag82] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. In *FOCS*, pages 32–39. IEEE Computer Society, 1982. (Cited on page 48.)

[LF06] Éric Levieil and Pierre-Alain Fouque. An Improved LPN Algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006. (Cited on page 81.)

[LHA+12] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 626–642. Springer, 2012. (Cited on page 7.)

[LLL82] A.K. Lenstra, H.W.jun. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982. (Cited on pages 50 and 119.)

[LM10] M. Lochter and J. Merkle. Elliptic curve cryptography (ECC) Brainpool standard curves and curve generation. RFC 5639 (Informational), March 2010. (Cited on page 102.)

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, 1996. (Cited on page 163.)

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by Enumeration: An Update. In Ed Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2013. (Cited on page 81.)

[LRT14] Duc-Phong Le, Matthieu Rivain, and Chik How Tan. On double exponentiation for securing rsa against fault analysis. In Josh Benaloh, editor, *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2014. (Cited on page 62.)

[MHER14] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, pages 1–12, 2014. (Cited on pages 48 and 63.)

[MHMP14] Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. *J. Cryptographic Engineering*, 4(1):33–45, 2014. (Cited on pages 81, 84, 88, and 89.)

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985. (Cited on pages 5 and 75.)

[MM99] Henry McKean and Victor Moll. *Elliptic curves: function theory, geometry, arithmetic*. Cambridge University Press, 1999. (Cited on page 91.)

[MMM04] C. McIvor, M. McLoone, and J.V. McCanny. Modified Montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings - Computers and Digital Techniques*, 151(6):402–408, 2004. (Cited on pages 39, 40, and 41.)

[MMNT13] Diana Maimut, Cédric Murdica, David Naccache, and Mehdi Tibouchi. Fault attacks on projective-to-affine coordinates conversion. In Emmanuel Prouff, editor, *COSADE*, volume 7864 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2013. (Cited on pages 81 and 82.)

[Möl04] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 335–351. Springer, 2004. (Cited on page 102.)

[Mon71] H. L. Montgomery. *Topics in multiplicative number theory*. Springer-Verlag, 1971. (Cited on page 67.)

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985. (Cited on pages 26 and 27.)

[MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 58–75. Springer, 2012. (Cited on page 48.)

[MOV96] Alfred Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. 1996. (Cited on page 126.)

[MS91] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect polynomial random number generators. *J. Cryptology*, 3(3):157–172, 1991. (Cited on pages 8, 115, 153, and 156.)

[MSPV07] Nele Mentens, Kazuo Sakiyama, Bart Preneel, and Ingrid Verbauwhede. Efficient pipelining for modular multiplication architectures in prime fields. In Hai Zhou, Enrico Macii, Zhiyuan Yan, and Yehia Massoud, editors, *ACM Great Lakes Symposium on VLSI*, pages 534–539. ACM, 2007. (Cited on pages 39, 40, 42, and 169.)

[Mur14] Cédric Murdica. *Physical Security of Elliptic Curve Cryptography*. PhD thesis, Télécom ParisTech, 2014. (Cited on page 99.)

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996. (Cited on page 8.)

[NMSiK01] Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shin ichi Kawamura. Implementation of rsa algorithm based on rns montgomery multiplication. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 364–376. Springer, 2001. (Cited on pages 39, 40, 42, 43, and 169.)

[NNTW05] David Naccache, Phong Q. Nguyen, Michael Tunstall, and Claire Whelan. Experimenting with Faults, Lattices and the DSA. In Serge Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2005. (Cited on pages 81, 82, 96, 97, and 98.)

[NS97] Phong Q. Nguyen and Jacques Stern. Merkle-hellman revisited: A cryptoanalysis of the qu-vanstone cryptosystem based on group factorizations. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 1997. (Cited on pages 50 and 53.)

[NS00] Phong Q. Nguyen and Jacques Stern. Lattice reduction in cryptology: An update. In Wieb Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 85–112. Springer, 2000. (Cited on page 50.)

[NS02] Phong Q. Nguyen and Igor Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *J. Cryptology*, 15(3):151–176, 2002. (Cited on pages 81, 94, and 120.)

[NS03] Phong Q. Nguyen and Igor Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003. (Cited on pages 81, 94, and 97.)

[NS04] Phong Q. Nguyen and Damien Stehlé. Low-Dimensional Lattice Basis Reduction Revisited. In Duncan A. Buell, editor, *ANTS*, volume 3076 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2004. (Cited on page 80.)

[NT12] Phong Q. Nguyen and Mehdi Tibouchi. Lattice-Based Fault Attacks on Signatures. In Joye and Tunstall [JT12], pages 201–220. (Cited on pages 81, 82, and 97.)

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003. (Cited on page 154.)

[OLARH14] Thomaz Oliveira, Julio López, Diego F. Aranha, and Francisco Rodríguez-Henríquez. Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014. (Cited on pages 80, 103, 106, 107, 111, 112, and 113.)

[Ora] Oracle. JavaCard 3.0.1 Platform Specification. `http://www.oracle.com/technetwork/java/javacard/overview/`. (Cited on pages 25 and 28.)

[Oru95] Holger Orup. Simplifying quotient determination in high-radix modular multiplication. In *IEEE Symposium on Computer Arithmetic*, pages 193–. IEEE Computer Society, 1995. (Cited on pages 41 and 42.)

[PJKL02] Young-Ho Park, Sangtae Jeong, Chang Han Kim, and Jongin Lim. An Alternate Decomposition of an Integer for Faster Point Multiplication on Certain Elliptic Curves. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2002. (Cited on pages 80, 82, 93, and 171.)

[Pol00] John M. Pollard. Kangaroos, monopoly and discrete logarithms. *J. Cryptology*, 13(4):437–447, 2000. (Cited on page 84.)

[PV04] Dan Page and Frederik Vercauteren. Fault and side-channel attacks on pairing based cryptography. *IACR Cryptology ePrint Archive*, 2004:283, 2004. (Cited on page 55.)

[Res00] Certicom Research. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, September 2000. Version 1.0. (Cited on page 88.)

[RG13] Pablo Rauzy and Sylvain Guilley. A formal proof of countermeasures against fault injection attacks on crt-rsa. *Journal of Cryptographic Engineering*, pages 1–13, 2013. (Cited on pages 48 and 63.)

[Riv09] Matthieu Rivain. Securing rsa against fault analysis by double addition chain exponentiation. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2009. (Cited on pages 25 and 62.)

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. (Cited on page 4.)

[S+14] W.A. Stein et al. *Sage Mathematics Software (Version 6.2)*. The Sage Development Team, 2014. http://www.sagemath.org. (Cited on pages 34, 50, and 53.)

[Sch00a] Werner Schindler. A timing attack against rsa with the chinese remainder theorem. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2000. (Cited on page 27.)

[Sch00b] Richard Schroeppel. Elliptic curves: Twice as fast! Presentation at the CRYPTO 2000 Rump Session, 2000. (Cited on page 106.)

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. (Cited on page 50.)

[SGF10] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. From program verification to program synthesis. In Manuel V. Hermenegildo and Jens Palsberg, editors, *POPL*, pages 313–326. ACM, 2010. (Cited on page 48.)

[Sha81] Adi Shamir. The generation of cryptographically strong pseudo-random sequences. In Allen Gersho, editor, *CRYPTO*, page 1. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981. (Cited on page 115.)

[Sha99] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999. US Patent 5,991,415. (Cited on page 25.)

[Sho06] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006. (Cited on page 162.)

[Sko10] Sergei Skorobogatov. Optical fault masking attacks. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 23–29. IEEE Computer Society, 2010. (Cited on page 39.)

[SLRBE05] Armando Solar-Lezama, Rodric M. Rabbah, Rastislav Bodík, and Kemal Ebcioglu. Programming by sketching for bit-streaming programs. In Vivek Sarkar and Mary W. Hall, editors, *PLDI*, pages 281–294. ACM, 2005. (Cited on page 48.)

[Smi13] Benjamin Smith. Families of Fast Elliptic Curves from $\mathbb{Q}$-curves. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2013. (Cited on page 80.)

[SS81] Richard Schroeppel and Adi Shamir. A $t=o(2^{n/2})$, $s=o(2^{n/4})$ algorithm for certain np-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981. (Cited on page 154.)

[Ste87] Jacques Stern. Secret linear congruential generators are not cryptographically secure. In *FOCS*, pages 421–426. IEEE Computer Society, 1987. (Cited on page 119.)

[Suz07] Daisuke Suzuki. How to maximize the potential of fpga resources for modular exponentiation. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2007. (Cited on pages 39, 40, and 42.)

[SvdW06] Andrew Shallue and Christiaan van de Woestijne. Construction of rational points on elliptic curves over finite fields. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006. (Cited on pages 103 and 104.)

[TcKK99] Alexandre F. Tenca and Çetin Kaya Koç. A scalable architecture for montgomery multiplication. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 1999. (Cited on pages 39, 40, 41, and 169.)

[TFHA+11] Jonathan Taverne, Armando Faz-Hernández, Diego F. Aranha, Francisco Rodríguez-Henríquez, Darrel Hankerson, and Julio López. Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. *J. Cryptographic Engineering*, 1(3):187–199, 2011. (Cited on page 111.)

[The] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. http://www.openssl.org/. (Cited on page 28.)

[Tib14] Mehdi Tibouchi. Elligator Squared: Uniform points on elliptic curves of prime order as uniform random strings. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography*, Lecture Notes in Computer Science. Springer, 2014. To appear. Available as http://eprint.iacr.org/2014/043. (Cited on pages 102, 103, 104, and 106.)

[TK10] Elena Trichina and Roman Korkikyan. Multi fault laser attacks on protected CRT-RSA. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 75–86. IEEE Computer Society, 2010. (Cited on page 61.)

[Tuc97] Allen B. Tucker, editor. *The Computer Science and Engineering Handbook*. CRC Press, 1997. (Cited on page 158.)

[Vig08] David Vigilant. Rsa with crt: A new cost-effective solution to thwart fault attacks. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008. (Cited on page 25.)

[Vin54] Ivan M. Vinogradov. *Elements of number theory*. Dover, 1954. (Cited on page 162.)

[Wal99] Colin D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 1999. (Cited on page 27.)

[WG12] Erich Wenger and Johann Großschädl. An 8-bit avr-based elliptic curve cryptographic risc processor for the internet of things. In *MICRO Workshops*, pages 39–46. IEEE Computer Society, 2012. (Cited on page 92.)

[Woo12] TD Wooley. Vinogradov's mean value theorem via efficient congruencing. *Annals of Mathematics*, 175(3):1575–1627, 2012. (Cited on pages 155 and 165.)

[WPF+10] Yi Wei, Yu Pei, Carlo A. Furia, Lucas Serpa Silva, Stefan Buchholz, Bertrand Meyer, and Andreas Zeller. Automated fixing of programs with contracts. In Paolo Tonella and Alessandro Orso, editors, *ISSTA*, pages 61–72. ACM, 2010. (Cited on page 49.)

[WWGH11] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium*. USENIX Association, 2011. (Cited on page 102.)

[WWY+12] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 109–120. ACM, 2012. (Cited on page 102.)

[YMH02] Sung-Ming Yen, Sang-Jae Moon, and JaeCheol Ha. Hardware fault attackon rsa with crt revisited. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2002. (Cited on page 21.)

[YY07] Adam L. Young and Moti Yung. Space-efficient kleptography without random oracles. In Teddy Furon, François Cayre, Gwenaël J. Doërr, and Patrick Bas, editors, *Information Hiding*, volume 4567 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2007. (Cited on page 102.)

[YY10] Adam L. Young and Moti Yung. Kleptography from standard assumptions and applications. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2010. (Cited on page 102.)

Author Index

Résumé

Dans cette thèse, nous nous intéressons à la sécurité de générateurs pseudo-aléatoires et d'implémentations de schémas de signature.

Concernant les schémas de signature, nous proposons, dans le cas d'une implémentation répandue de RSA, différentes attaques par injection de faute effectives quelque soit l'encodage du message. Nous présentons par ailleurs une contre-mesure infective prouvée sûre pour protéger le schéma RSA–PSS contre un certain nombre de fautes non aléatoires. Nous étudions également le schéma ECDSA couplé aux techniques d'accélération GLV/GLS. En fonction des implémentations, nous prouvons soit la bonne distribution du nonce utilisé, soit qu'il présente un biais permettant une attaque. Enfin, nous élaborons un outil qui recherche automatiquement des attaques par faute à partir d'une implémentation et d'une politique de faute, outil appliqué avec succès sur des implémentations de RSA et de ECDSA.

Concernant les générateurs pseudo-aléatoires algébriques, nous étudions les générateurs non-linéaires et améliorons certaines attaques en diminuant l'information donnée à l'adversaire. Nous nous intéressons également à la sécurité du générateur Micali-Schnorr à travers quelques attaques et une étude statistique de son hypothèse de sécurité. Finalement nous proposons une cryptanalyse de tout schéma à clé publique basé sur la factorisation ou le logarithme discret dont la clé secrète est générée à partir d'un générateur linéaire.

Abstract

In this thesis, we are interested in the security of pseudorandom number generators and of implementations of signature schemes.

Regarding the signature schemes, we propose, in the case of a widespread implementation of RSA, various fault attacks which apply to any padding function. In addition we present a proven secure infective countermeasure to protect the RSA–PSS scheme against some non-random faults. Furthermore we study the ECDSA scheme coupled with the GLV/GLS speed-up techniques. Depending on the implementations, we prove either the good distribution of the used nonce, or that it has a bias, thereby enabling an attack. Finally we develop a tool for automatically finding fault attacks given an implementation and a fault policy, which is successfully applied to some RSA and ECDSA implementations.

Regarding pseudorandom number generators, we study the nonlinear ones and improve some attacks by reducing the information available to the adversary. We also are interested in the security of the Micali-Schnorr generator through various attacks and a statistical study of its security assumption. Finally we propose a cryptanalysis of any public-key scheme based on the factorization or the discrete logarithm when the secret key is generated using a linear generator.